

Nederlandse organisatie
voor toegepast
natuurwetenschappelijk
onderzoek



Fysisch en Elektronisch
Laboratorium TNO



DTIC FILE COPY

83-3207

rapport no.
FEL-89-A267

exemplaar no.

7

Kwaliteit van Expertsystemen: Methoden en
Technieken

AD-A225 059

DTIC
ELECTE
AUG 03 1990
S E D

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

90 08 03 050

TNO-rapport



Postbus 96864
2509 JG 's-Gravenhage
Oude Waalsdorperweg 63
's-Gravenhage

Telefoon 070 - 26 42 21

exemplaar no.

titel

7

Kwaliteit van Expertsystemen: Methoden en Technieken

Indien dit rapport in opdracht werd uitgebracht, wordt voor de rechten en verplichtingen van opdrachtgever en opdrachtnemer verwezen naar de 'Algemene Voorwaarden voor Onderzoeksopdrachten TNO', dan wel de betreffende terzake tussen partijen gesloten overeenkomst.

§ TNO

auteur(s):

Drs. J.H.J. Lenting (RL)

Drs. M. Perre (FEL-TNO)

[illegible]

rubricering

titel

samenvatting

rapport

: ongerubriceerd

: ongerubriceerd

: ongerubriceerd

oplage

aantal bladzijden

aantal bijlagen

: 40

:77

: -

datum

: december 1989



DTIC
ELECTE
AUG 03 1990

E

D

rapport no. : FEL-89-A267
titel : Kwaliteit van Expertsystemen: Methoden en Technieken

auteur(s) : Drs. J.H.J. Lenting en Drs. M. Perre
instituut : Fysisch en Elektronisch Laboratorium TNO

datum : december 1989
hdo-opdr.no. : A89K602
no. in iwp '89 : 704.2

=====

SAMENVATTING

Dit rapport is het resultaat van de tweede fase van het technologie-project "Kwaliteit van Expertsystemen", uitgevoerd in opdracht van het Ministerie van Defensie, Directoraat-Generaal Wetenschappelijk Onderzoek en Ontwikkeling. Deelnemers aan het project zijn het Fysisch en Elektronisch Laboratorium TNO (FEL-TNO), de Rijksuniversiteit Limburg (RL) en het Research Instituut voor Kennis-Systemen (RIKS). In dit rapport wordt, uitgaand van [FEL1989-148], dieper ingegaan op het probleem van de kwaliteitsbeheersing bij kennissystemen. Ten eerste wordt er een methode gepresenteerd die de ontwikkeling van kennissystemen beschouwd als een (parallele) activatie van modelleringsprocessen. Ten tweede is er aandacht voor de conceptuele overeenkomsten tussen gegevensbanken en kennisbanken. Implicatie hiervan is het gebruik van Nijssens Informatieanalyse Methode (NIAM) als specificatiemethode voor kennisbanken. Ten derde is het onderwerp modulariteit en structurering van kennisbanken onderzocht met daarnaast aandacht voor de toepasbaarheid van conventionele testmethodologieën op kennissysteemsoftware. Ten slotte wordt aan de hand van een voorbeeld aangetoond dat met de integratie van gegevensbanktheorie en kunstmatige intelligentie een belangrijke stap kan worden gezet op het pad naar een betere kwaliteitsbeheersing van kennissystemen.

report no. : FEL-89-A267
title : Quality of Expert Systems: Methods and Techniques

author(s) : J.H.J. Lenting MA and M. Perre MA
institute : TNO Physics and Electronics Laboratory

date : December 1989
NDRO no. : A89K602
no. in pow '89 : 704.2

=====

ABSTRACT

This report is the result of the second phase of the technology project "Quality of Expert Systems", carried out under commission of the Ministry of Defence, Director Defence Research and Development. Participants in the project are TNO Physics and Electronics Laboratory (FEL-TNO), University of Limburg (RL) and the Research Institute for Knowledge Systems (RIKS). Based on [FEL1989-148] this report examines more thoroughly the problems that are related to the quality of expert systems. Firstly a method is presented which views the development of expert systems as a (parallel) activation of modelling processes. Secondly the conceptual similarities between databases and knowledgebases are stressed. As an implication of this the use of Nijssens Information Analysis Method as a knowledgebase specification method is proposed. Thirdly the modularity and structure of knowledge bases is examined, together with the applicability of conventional testing methodologies in expert systems. Lastly it is demonstrated with an example that the integration of database theory and artificial intelligence signifies a step in the direction of a better quality control of expert systems

	SAMENVATTING	1
	ABSTRACT	2
	INHOUDSOPGAVE	3
1	INLEIDING	5
2	KWALITEITSCRITERIA	7
3	KWALITEIT VAN HET ONTWIKKELPROCES	12
3.1	Inleiding	12
3.2	Conventionele kijk op onconventionele ontwikkeling	13
3.3	Levenscyclus van kennissysteem-ontwikkeling	14
4	KWALITEIT VAN DE SPECIFICATIE	18
4.1	Inleiding	18
4.2	Kwaliteitscriteria en -eisen	19
4.3	Een specificatiemethode voor kennissystemen	23
4.3.1	Kennisanalyse met NIAM	23
4.3.2	Beperkingsregels in NIAM	26
4.4	Beperkingsregels in database-management-systemen	30
4.5	Van database naar knowledgebase	33
4.6	Gestructureerde kennismodellen in slecht gestructureerde domeinen	40
5	KWALITEIT VAN HET PRODUKT	42
5.1	Inleiding	42
5.2	Testbaarheid van kennissystemen	44
5.2.1	Conventionele testmethodologie	44
5.2.2	Toepasbaarheid van conventionele testmethodologie op kennissysteem-software	47

5.3	Structurering en modulariteit	50
5.3.1	Modulariteit in conventionele software	50
5.3.2	Modulaire structurering van kennisbanken	53
5.3.3	Cohesie en koppeling bij kennisbanken	56
5.4	Database-technologie en kwaliteitsbeheersing	57
5.4.1	Integriteit in databases en knowledgebases	57
5.4.2	Integriteit in knowledgebase-management-systemen	60
5.4.3	POSTGRES als knowledgebase-management-systeem	61
6	CONCLUSIES EN AANBEVELINGEN	71
7	REFERENTIES	74

1 INLEIDING

Dit rapport is het vervolg op de oriënterende studie [FEL1989-148]. Als zodanig past het in hetzelfde kader, te weten dat van de kwaliteit van kennissystemen, waarbij in de ontwikkeling van kennissystemen drie aspecten worden onderscheiden: specificatie, proces en produkt.

De inhoud van dit rapport is in vergelijking met het vorige op drie gebieden nader toegespitst. In de eerste plaats komt het probleem van de kwaliteits-bepaling in dit rapport niet of nauwelijks aan de orde. Hieraan is al vrij uitgebreid aandacht besteed in [FEL1989-148]. Hoewel een verdere uitdieping van de in dit kader besproken evaluatie-problematiek mogelijk is wordt het zinvoller geacht de nadruk te leggen op *constructieve* methoden en technieken, die tot kennissystemen van goede (lees: betere) kwaliteit kunnen leiden. Anders gezegd, dit rapport gaat over kwaliteits-beheersing.

Daarnaast spitst het rapport zich toe op dat onderdeel van de systeemontwikkeling waarin kennissystemen het duidelijkst verschillen van andere software: de specificatie en implementatie van de *kennisbank*. De opkomst van kennissystemen als commercieel produkt is hoofdzakelijk te danken aan het in de vorm van expertsysteem-shells geconcretiseerde principe van hergebruik van software. Ook bij anderssoortige software, bijvoorbeeld relationele database-management-systemen (RDBMS'en) heeft dit principe zijn vruchten afgeworpen. In dit rapport zal dan ook kennisysteemontwikkeling primair beschouwd worden als kennisbank-onwikkeling, d.w.z. er wordt van uit gegaan dat software hulpmiddelen (compilers voor een programmeertaal, expertsysteem-shells of database-management-systemen) voorhanden zijn en foutvrij werken.

Tenslotte is er prioriteit gegeven aan die criteria waaraan kennissystemen - afgaande op de in [FEL1989-148] besproken literatuur - het slechtst lijken te voldoen. Op basis van de uitwerking van die criteria in (min of meer) meetbare 'maten' zijn de methoden en technieken geselecteerd, die het beste perspectief bieden wat betreft kwaliteitsbeheersing van kennissystemen. Vele hiervan zijn gebaseerd op of zelfs regelrecht ontleend aan (relationele) database-technologie. Hoewel het vooralsnog niet 'bewezen' is dat een relationele-database-aanpak voor *alle* typen van kennissystemen adequaat is, lijkt het dat

deze aanpak op het vlak van kwaliteitsbeheersing een beter perspectief biedt dan andere, in [FEL1989-148] besproken, ontwikkelmethodologieën. Omdat de begrippen 'shell' en 'kennisbank' in de praktijk op velerlei wijze geconcretiseerd zijn is zeer uiteenlopende software van het etiket 'kennissysteem' voorzien. Derhalve is het niet mogelijk - met behoud van 'algemene geldigheid' - scherpe definities van deze begrippen te geven. Voorlopig gelieve de lezer bij 'kennisbank' te denken aan een verzameling 'feiten' en 'afleidingsregels' en bij 'shell' aan software die in staat is op basis van een kennisbank vragen van de gebruiker te beantwoorden. Het gehanteerde kwaliteitsraamwerk in termen van specificatie, proces en produkt ([FEL1989-148]) is in dit rapport aangehouden, met dien verstande dat 'het proces' in dit rapport refereert aan het *overkoepelende* proces van kennissysteemontwikkeling (de life-cycle, dus), terwijl 'het produkt' en 'de specificatie' betrekking hebben op *onderdelen* daarvan, te weten de kennisbank en het kennismodel als specificatie daarvan. Het 'deelproces' van de implementatie van een kennisbank volgens een kennismodel wordt niet apart besproken, maar gerelateerd aan de voorgestelde vormen van specificatie en produkt.

Het resterende gedeelte van dit rapport bestaat uit vijf hoofdstukken. Allereerst wordt een overzicht gegeven van criteria die een rol spelen bij de kwaliteitsbeheersing van kennissystemen. Langs de lijnen van het kwaliteitsraamwerk worden in de drie volgende hoofdstukken respectievelijk de methoden en technieken voor de kwaliteitsbeheersing van het ontwikkelproces, de specificatie(s) en het produkt behandeld. Daarna volgt een hoofdstuk met conclusies en aanbevelingen. Ten slotte zijn de referenties opgenomen.

2 KWALITEITSCRITERIA

In paragraaf 4.2.2.3 van het fase 1-rapport [FEL1989-148], is de onderstaande, door Llinas en Rizzi [Llinas87] opgestelde lijst met test- en evaluatiecriteria gepresenteerd:

- Consistentie en Volledigheid van de kennisbank
- Software Quality criteria
- Mens/machine interactie
- Kosten
- Kwaliteit van uitleg
- Relevantie
- Betrouwbaarheid
- Portabiliteit
- Bijdrage aan Organisationele doelen
- Kwaliteit en nauwkeurigheid van beslissingen/advies/aanbevelingen
- Correctheid in redeneren
- Systeem-efficiëntie
- Aanpassingsvermogen (onderhoudbaarheid)
- Run-tijd
- Computationale efficiëntie
- Intelligent gedrag

Deze lijst is echter geen goed uitgangspunt, omdat er teveel vage en overlappende items in voorkomen. Bovendien vallen sommige van de genoemde criteria (consistentie en volledigheid, run-tijd) eerder onder de noemer 'maten', terwijl andere (software quality criteria, kosten, bijdrage aan organisationele doelen) wel een zeer triviale 'uitwerking' zijn van de bijbehorende filosofieën.

In dit rapport wordt uitgegaan van een eigen indeling in criteria en uitwerkingen daarvan (maten). Aan metriecken (kwantifikaties van die maten) wordt in dit rapport hoegenaamd geen aandacht besteed. Dit onderwerp is in [FEL1989-148] al tamelijk uitgebreid behandeld. Weliswaar heeft deze behandeling meer het karakter van een bespreking van een aantal min of meer ad hoc geselecteerde metriecken dan van een evaluatie-handboek

voor kennissystemen, maar het presenteren van een dergelijk handboek zou eigenlijk een vorm van boerenbedrog zijn. Het onderwerp kennissystemen is te breed om er een zinvol evaluatie-handboek (in de vorm van een beslisboom met metriecken als bladeren) voor te fabriceren. De in de ESPRIT-proceedings van 1987 vermelde 'teleurstellende' resultaten van een zeer groot opgezet project voor het vaststellen van geschikte software-metriecken spreken wat dit betreft boekdelen! Onderwerp van dit rapport is de kwaliteitsbeheersing en meer in het bijzonder de methoden en technieken die tot een produkt van betere kwaliteit kunnen leiden.

Hieronder is deze visie op kwaliteit in schematische vorm weergegeven. Kwaliteitsaspecten van specificatie, proces en produkt zijn apart vermeld om de interpretatie van de gebruikte termen te vergemakkelijken. Enkele termen verdienen wellicht desondanks enige toelichting. De termen consistentie en volledigheid worden in de literatuur, mede afhankelijk van de context, voor tamelijk verschillende begrippen gebruikt. Een scherp onderscheid tussen de verschillende betekenissen zal in Hoofdstuk 5 gemaakt worden.

Met falsifieerbaarheid van specificaties wordt gerefereerd aan het feit dat 'specificaties' die dermate vaag zijn dat zelfs zeer uiteenlopende implementaties er aan 'geverifieerd' kunnen worden, niet veel zin hebben. Bij robuustheid van het ontwikkelproces moet men denken aan ontsnappingsclausules die anticiperen op onvoorziene tegenslagen en deze zo goed mogelijke opvangen. Robuustheid van het produkt heeft betrekking op beveiliging tegen onverwachte externe invloeden. Deze kunnen zowel menselijk als technisch van aard zijn. Fail-safe slaat op het opvangen van technische storingen als stroomuitval, monkey-proof op het bestand zijn tegen bijvoorbeeld menselijk wangedrag achter het toetsenbord. Met aansluiting van het produkt op de specificatie wordt bedoeld dat de afstand tussen specificatie en produkt niet te groot mag zijn. Met name voor kennissystemen wordt nogal eens gepleit voor specificatie 'op het knowledge level' [Newell82,Breuker87] op grote 'afstand' van de implementatie. Dat kan op zich zeer voordelig zijn mits die afstand wel op geformaliseerde wijze overbrugd kan worden via een (aantal) veel dichterbij de implementatie staande specificatie(s).

De layout van de schema's geeft de hiërarchie van de concepten aan. Zo worden bruikbaarheid en aanpasbaarheid als de belangrijkste kwaliteitsaspecten van specificaties gezien, waarbij binnen bruikbaarheid betrouwbaarheid en efficiëntie worden onderscheiden. Voor een beter begrip is het goed zich te realiseren dat sommige (de meeste) van de in het schema vermelde criteria en maten desiderata van specificatie c.q. proces c.q. produkt zijn terwijl andere desiderata van de hiertoe angewende technieken zijn. Tot deze laatste categorie behoren bijvoorbeeld ontwerp- en ontwikkel-gemak bij specificaties. De kwaliteitsaspecten waaraan we de meeste aandacht zullen besteden in dit rapport zijn aangekruist. Een meer gedetailleerde toelichting op de selectie van juist deze aspecten als sleutelbegrippen voor kwaliteit van specificatie en produkt zal gegeven worden in de betreffende hoofdstukken van dit rapport.

Kwaliteit van het ontwikkelproces

	bruikbaarheid			
		betrouwbaarheid		
			robuustheid	(x)
			controleerbaarheid	(x)
		efficiëntie		
			taakopdeling	(x)
			taaktoewijzing	
			capaciteiten	
			overdraagbaarheid	(x)

Figuur 2.1: Kwaliteit van het ontwikkelproces.

Kwaliteit van de specificatie

	bruikbaarheid		
		betrouwbaarheid	
			integriteit
			consistentie (x)
			volledigheid (x)
			testbaarheid
			modulariteit (x)
			falsifieerbaarheid (x)
			efficiëntie
			ontwerpgemak
			ontwikkelgemak
	aanpasbaarheid		
		onderhoudbaarheid	
			leesbaarheid
			modulariteit
		overdraagbaarheid	
			apparatuur onafhankelijk
			implementatie-onafhankelijk

Figuur 2.2: Kwaliteit van de specificatie.

Kwaliteit van het produkt

	bruikbaarheid		
		betrouwbaarheid	
			integriteit
			consistentie (x)
			volledigheid (x)
			correctheid
			robuustheid
			'fail-safe' c.q. 'monkey-proof'
			graceful degradation
			testbaarheid
			modulariteit (x)
			aansluiting op specificatie (x)
		efficiëntie	
			computationele efficiëntie
			gebruikersvriendelijkheid
	aanpasbaarheid		
		onderhoudbaarheid	
			leesbaarheid
			modulariteit (x)
	overdraagbaarheid		
		apparatuur onafhankelijk	
	integreerbaarheid (koppeling aan 'andere' software)		

Figuur 2.3: Kwaliteit van het produkt.

3 KWALITEIT VAN HET ONTWIKKELPROCES

3.1 Inleiding

Voor een beoordeling van de kwaliteit van het ontwikkelproces zijn in het vorige hoofdstuk een aantal criteria genoemd die hieronder nogmaals zijn weergegeven. De belangrijkste (in het kader van dit rapport) zijn voorzien van een kruisje.

Kwaliteit van het ontwikkelproces

	bruikbaarheid		
		betrouwbaarheid	
			robuustheid (x)
			controleerbaarheid (x)
		efficiëntie	
			taakopdeling (x)
			taaktoewijzing
			capaciteiten
			overdraagbaarheid (x)

Figuur 3.1: Kwaliteit van het ontwikkelproces.

In hoofdstuk 4 van het fase 1-rapport [FEL1989-148] zijn een aantal punten genoemd die in het oog moeten worden gehouden bij het hanteren van een bepaalde ontwikkelmethode. Een indeling in fasen met goed gedefinieerde eindprodukten en het ondersteunen van een iteratief ontwikkelproces zijn daarvan de belangrijkste.

Wat betreft het ontwikkelproces is het zinvol om een onderscheid te maken tussen drie typen systemen: transactie-verwerkende systemen (TVS), beslissingsondersteunende systemen (BOS) en kennissystemen (KS). (Het gaat nu niet om een scherpe afgrenzing;

kennissystemen kunnen namelijk ook een beslissing ondersteunen). In een TVS wordt veelal routinematig werk verricht, gebaseerd op vaste formulieren en procedures, zoals bij administratieve toepassingen. De problemen zijn goed gestructureerd, wat tot een min of meer 'soepele' informatiebehoefte-analyse leidt. Desondanks zal het ontwikkelen van TVS'en niet zo triviaal zijn dat van een 'puur' sequentieel ontwikkelproces kan worden gesproken. Een methodiek als de System Development Methodology (SDM), zoals besproken in [FEL1989-148], biedt voldoende faciliteiten om een TVS te ontwikkelen. Bij een BOS gaat het om minder goed gestructureerde problemen, die men door middel van wiskundige en statistische modellen tracht op te lossen. Hier zijn de gebruikersbehoefte al veel minder duidelijk dan bij TVS'en het geval is. Om te kunnen komen tot volledige en consistente specificaties van een BOS, zal samen met de gebruiker een sterk iteratief ontwikkelpad gevolgd moeten worden. Prototyping is in dit geval een goede oplossing. Bij de ontwikkeling van een KS speelt 'kennis' een belangrijke rol. De problemen die hier worden aangepakt kenmerken zich door het feit dat er geen bekend algoritme is om ze op te lossen. In een relatief moeizaam proces van kennisverwerving moet worden getracht structuur aan te brengen in de kennis die een expert heeft op een bepaald gebied. Dit leidt ertoe dat men in een ontwikkelproces van een KS te maken heeft met een grote onzekerheid. Een iteratief proces, gebaseerd op prototyping lijkt de aangewezen aanpak. In het verleden betekende dit echter meestal dat er geen analyse van probleemdomen werd gemaakt, zoals bij meer conventionele systemen de gewoonte is. SKE is een ontwikkelmethode die het opstellen van specificaties juist benadrukt. Uit de volgende hoofdstukken zal blijken dat ook in dit rapport deze aanpak wordt gevolgd, zij het dat de specificatie-methode niet KADS is, maar NIAM.

3.2 Conventionele kijk op onconventionele ontwikkeling

In paragraaf 4.2.1 van het fase 1-rapport [FEL1989-148] zijn een aantal AI-ontwikkelmethoden de revue gepasseerd. Hieruit bleek dat in de meeste methoden de nadruk werd gelegd op de kennisverwervings-fase. Dit is gezien de intrinsieke moeilijkheidsgraad van deze activiteit niet zo verwonderlijk. Alleen blijft de ontwikkeling van een kennisstelsel niet tot deze fase beperkt. Met name de 'Structured Knowledge Engineering'-methode (SKE) doet, gebaseerd op SDM, een (zij het beknopte) uitspraak over de inrichting van de overige fasen, zoals technisch ontwerp,

implementatie, testen, acceptatie en onderhoud. Het voordeel van de SKE-aanpak is dat men terugvalt op een 'bekende conventionele' ontwikkelmethode, i.c. SDM: dit vergroot de kans van toepassing.

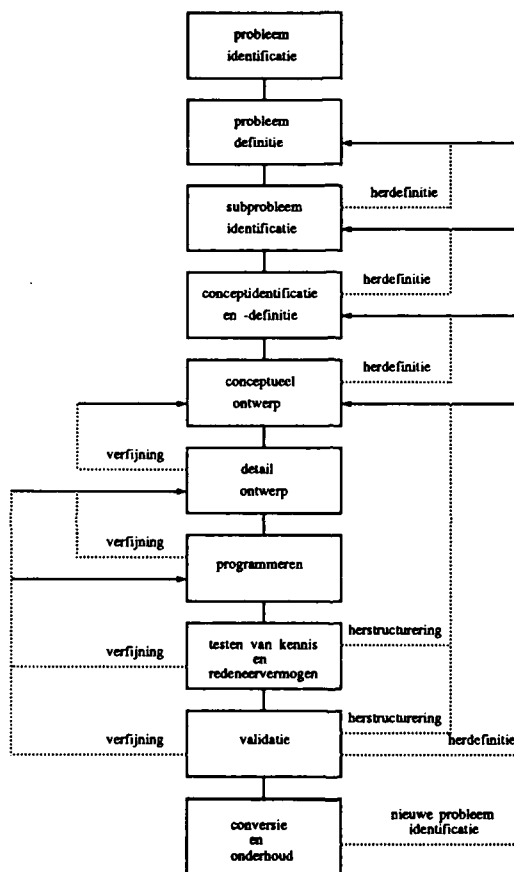
In het navolgende gaan we er van uit dat de fasen acceptatie en integratie niet werkelijk zullen verschillen van wat bij conventionele systeemontwikkeling bekend is. Dit betekent *niet* dat deze fasen minder belangrijk zijn. De nadruk zal moeten liggen op de fasen kennis- en informatiebehoeften-analyse tot en met de implementatie van het kennissysteem.

3.3 Levenscyclus van kennissysteem-ontwikkeling

Men verwacht van een ontwikkelmethode voor kennissystemen dat zij een goede fasering heeft, leidt tot duidelijk gedefinieerde tussen- en eindprodukten, iteratief kan worden toegepast, ruimte biedt voor prototyping en voorziet in een welomschreven test- en evaluatiefase. Hoewel dit een nogal ambitieuze opsomming is, lijkt de methode van Kerschberg c.s. een goede kandidaat te zijn [Kerschberg89]. Een belangrijk punt is dat deze auteur 'op kennis gebaseerde systemen' ziet als een evolutionaire ontwikkeling van conventionele systemen. Het is dan ook niet zo verwonderlijk dat hij een van de grote voorstanders is van de zogenaamde 'expert database systemen', die een samensmelting zijn van database- en expertsystemen.

Deze ontwikkelmethode steunt voor een aanzienlijk deel op technieken als prototyping, *zonder* dat de ontwikkeling van een conceptueel model van het probleemdomein achterwege blijft. Om niet in een sequentiële ontwikkeling te blijven steken, bestaan de afzonderlijke fasen van deze methode uit 'processen' die kunnen worden *geactiveerd*, *gede-activeerd* en *gere-activeerd*. Stelregel is dat een proces pas kan worden geactiveerd als het voorgaande proces minstens éénmaal eerder is geactiveerd (met uitzondering van het *allereerste* proces natuurlijk). Een *actief* proces kan elk *gede-activeerd* proces *re-activeren*.

De processen, ofwel fasen, zijn weergegeven in figuur 3.2. Er is uitgebreide aandacht voor activiteiten als 'verfijnen', 'herdefiniëren' en 'herstructureren'. Het eerste proces, *probleem-identificatie*, heeft betrekking op de eerste verkenning van het onderhavige



Figuur 3.2: Levenscyclus van kennisysteem-ontwikkeling.

probleem. Er wordt o.a. vastgesteld wie als expert is te beschouwen en met welk type probleem men eigenlijk te maken heeft.

Vervolgens wordt in het *probleemdefinitie*-proces een kennis- en informatiebehoefte-analyse uitgevoerd. In nauwe samenwerking met een expert wordt een informele karakterisering gemaakt van de begrippen en concepten die op het probleemgebied een rol spelen. De vraag welk doel het te ontwikkelen systeem moet dienen (interpretatie van satelliet-gegevens, weergave van gegevens in een 'head-up display' van een vliegtuig, 'archivering' van zeer schaarse kennis) wordt beantwoord. In deze fase is het ook de bedoeling om de uitvoerbaarheid van het project te beoordelen. Dit zal meestal geschieden aan de hand van technische en economische factoren. Het valt te verwachten dat dit proces grote kans maakt op 'latere' re-activatie, in het bijzonder wanneer meer gegevens 'boven water' komen.

Het proces *subprobleem-identificatie* zorgt voor een opdeling van het probleem in 'handelbare' stukken. Dit leidt tot een verdere reductie van de risico's, omdat het nu mogelijk is om voor elk subprobleem in een latere fase een klein prototype te ontwikkelen, die vervolgens weer geïntegreerd zullen worden.

In de *concept-identificatie* en *-definitie* worden de verschillende concepten en hun onderlinge relaties vastgelegd die van belang zijn bij het oplossen van problemen op een bepaald gebied. Dit proces houdt o.a. in dat alle objecten, attributen, relaties en beperkingsregels worden gemodelleerd. Hier wordt de basis gelegd voor de knowledgebase; wat hier gebeurt is het opstellen van een conceptueel model à la Nijssens Informatie-Analyse Methode (NIAM), waarover in het volgende hoofdstuk meer wordt genoemd. Technieken voor kennisverwerving vinden in dit proces ook hun plaats.

In het *conceptueel ontwerp* worden logische keuzes gemaakt ten aanzien van kennisrepresentatie en databases. De gedachte dat een AI-systeem zorgt voor de méérwaarde van een conventioneel systeem doet hier opgeld: de knowledgebase wordt gepartitioneerd in 'data-based' en 'rule-based' componenten.

Het *detail-ontwerp* definieert het fysieke systeemontwerp. Hier worden o.a. beschrijvingen en pseudo-code van programma's geproduceerd. Het komt dus neer op

een mapping van de belangrijkste concepten, relaties, informatiestromen en oplossingsstrategieën in een formeel representatie-raamwerk. Deze formalisatie-stap resulteert uiteindelijk in een prototype dat gebouwd wordt met behulp van een programmeertaal of 'expertsysteem shell'.

De werkelijke productie van het prototype vindt plaats in het *programmeer*-proces. Hier wordt het conceptuele ontwerp omgezet in een 'draaibaar' equivalent. Een geregelde terugkoppeling met het voorgaande proces valt te verwachten.

Vervolgens komt het proces waarin de *kennis* en het *redeneervermogen* van het prototype wordt *getest*. In nauwe samenwerking met de expert wordt het systeem aan de tand gevoeld. Aperte onjuistheden tracht men te corrigeren.

Tenslotte het *validatie*-proces. Nu wordt het systeem geconfronteerd met een groot aantal cases op het betreffende probleemgebied. De prestaties van het systeem worden vergeleken met die van de menselijke expert.

Bij de methode van Kerschberg c.s. is de mogelijkheid aanwezig om de ontwikkeling van een kennissysteem te integreren binnen een meeromvattend conventioneel systeem. De voorgestelde aanpak zou als een 'aparte' fase binnen een SDM-achtige faseringsmethode kunnen worden beschouwd. Conventionele en onconventionele systeemontwikkeling gaan hier hand in hand.

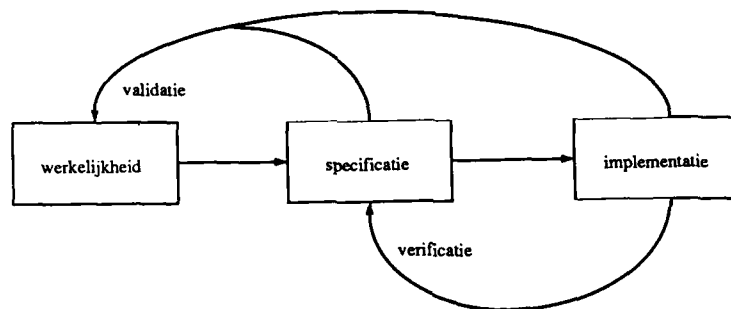
4 KWALITEIT VAN DE SPECIFICATIE

4.1 Inleiding

De kwaliteit van kennissystemen wordt in het algemeen als teleurstellend betiteld, zeker in vergelijking met meer conventionele software. Als problematische aspecten worden daarbij met name de kennisacquisitie, het testen/evalueren en het onderhoud van knowledgebases genoemd. Vooralsnog bestaat er echter weinig eensgezindheid over de wijze waarop deze problemen dienen te worden aangepakt. Een belangrijke motivatie voor het onderzoek is het feit dat binnen conventionele systemen steeds vaker 'intelligente' modules worden opgenomen, zonder dat kan worden voldaan aan dezelfde strikte kwaliteitseisen die worden gesteld aan het conventionele gedeelte. Dit geldt eens te meer wanneer het kritische toepassingen betreft, zoals procesbewakingssystemen in de chemische industrie of energiecentrales. Hetzelfde argument gaat ook op voor militaire commandovoeringssystemen, die in het jargon veelal worden aangeduid met de term C³I-systemen (Command, Control, Communications and Intelligence) [Napheys89]. Kenmerkend voor deze systemen is dat ze grote databases bevatten, waarmee de inzet van mensen en materieel wordt geleid. Door toevoeging van een redeneercomponent aan database-systemen kan de overgang naar zogeheten knowledgebase-systemen vergemakkelijkt worden. Hiermee wordt tevens bereikt dat faciliteiten op de gebieden integrity, concurrency, security en recovery in deze systemen kunnen worden opgenomen. In dit hoofdstuk wordt aangetoond dat een integratie van methoden en technieken uit de gegevensbanktheorie en de kunstmatige intelligentie de kwaliteitsbeheersing van kennissystemen ten goede kan komen.

Doel van dit hoofdstuk is de introductie van een methode om specificaties van een kennissysteem consistent en volledig vast te leggen. Er wordt een parallel getrokken met ontwikkelingen op het gebied van databases. Juist hier is in het verleden al de nodige aandacht besteed aan het integriteitsprobleem. De oplossingen die zijn gekozen op het vlak van specificatiemethoden en implementatietechnieken kunnen ook bij knowledgebases worden gebruikt.

criteria aan te wijzen die van belang zijn bij de beoordeling van de kwaliteit van specificaties. In het verleden is al naar voren gekomen dat de ontwikkeling van een systeem globaal is weer te geven als in figuur 4.2.



Figuur 4.2: Globale structuur van systeemontwikkeling.

Hieraan kunnen de validatie- en verificatie-activiteiten worden gekoppeld:

- specificatie en implementatie worden elk gevalideerd ten opzichte van de werkelijkheid,
- implementatie wordt geverifieerd ten opzichte van de specificatie.

Het kwaliteitsraamwerk is in deze voorstelling eenvoudig te herkennen. Immers, we zien de specificaties, het produkt en een (rudimentaire) weergave van het ontwikkelproces. Nu terug naar de lijst met kwaliteitscriteria. Elk van de 'hoofd'-criteria, bruikbaarheid en aanpasbaarheid, moet in de keuze vertegenwoordigd zijn.

De wijze waarop een bepaald kennisdomein van een kennisysteem gespecificeerd is verschilt niet wezenlijk van de specificatie die toegepast wordt bij de ontwikkeling van een gegevensbanksysteem. Het komt er op neer dat de kennisbank, als centraal onderdeel

van een kennissysteem, een modelmatige neerslag van een kennisdomein is. De betekenis van de kennis die in een kennisbank zit moet eenduidig in de specificatie zijn vastgelegd. Er wordt als het ware een 'conceptueel schema' ontworpen van de kennis, een 'semantische definitie'.

De eisen die hier aan specificaties worden gesteld, gebaseerd op [Wintraecken85], vloeien voort uit deze visie. Meer concreet betekent dit dat specificaties moeten voldoen aan het:

- conceptueel principe,
- 100% principe,
- natuurlijke taal principe.

Allereerst wordt voor elk van deze principes aangegeven wat ze betekenen.

Conceptueel principe

De grammatica beschrijft uitsluitend de conceptuele aspecten van de informatie-uitwisseling met het kennissysteem:

- alleen de betekenis van gegevens beschrijven
- geen realisatieaspecten, d.w.z implementatie onafhankelijk
- geen beschrijving interne/externe representatievorm
- geeft aan wat uitsluitend door de grammatica moet worden beschreven

In het laatste aandachtspunt is gesproken over een grammatica. Dit is een beschrijving van alle regels die voorschrijven welke toestanden en toestandsvergangen zich mogen voordoen en wat de betekenis van de in de kennisbank bewaarde gegevens is. De grammatica bestaat uit een specificatie van :

- de elementaire zinstypen waartoe de elementaire dieptestructuurzinnen dienen te behoren, die de inhoud van de kennisbank beschrijven (lot/nolot/idee/brug); dit betekent ook gebruik maken van de begrippen object, relatie en beperkingsregel
- de beperkingsregels (grafisch/niet-grafisch)

100% Principe

De grammatica beschrijft alle conceptuele aspecten van de kennis-uitwisseling met het kennissysteem:

- de betekenis van alle gegevens is beschreven in de grammatica.
- er worden geen andere dan de beschreven gegevens met het kennissysteem uitgewisseld.
- gebruikers mogen bij interpretatie van gegevens alleen de regels uit de grammatica gebruiken.
- nooit uitgaan van kennis over de werkelijkheid die impliciet bij gebruikers aanwezig is; alles expliciet beschrijven.

Natuurlijke taal principe

Kennis kan altijd worden beschreven d.m.v. elementaire dieptestructuurzinnen van een natuurlijke taal:

- de grammatica wordt beschreven d.m.v. natuurlijke taal
- alle impliciete kennis is expliciet gemaakt (--> elementaire dieptestructuurzinnen).

Het conceptueel principe draagt zorg voor de consistentie en volledigheid van specificaties. De volledigheid is gewaarborgd door het 100% principe. Het gebruik van een formele taal voor vastlegging van de specificaties leidt tot beknoptheid. Desondanks is de leesbaarheid 'goed', omdat het natuurlijke taal principe wordt gehanteerd. Overdraagbaarheid impliceert onafhankelijkheid van implementatie-zaken. Hier is ook het conceptueel principe van belang. Blijft over de aanpasbaarheid. Conceptueel en 100% principe zijn hiervoor voorwaarden.

Omdat binnen conventionele systemen op den duur gebruik gemaakt zal worden van 'kennis-modules', is het interessant om eens te kijken naar de mogelijkheden die conventionele methoden voor analyse en ontwerp van systemen bieden voor het specificeren van deze modules. Het komt de integreerbaarheid natuurlijk wel ten goede.

4.3 Een specificatiemethode voor kennissystemen

4.3.1 Kennisanalyse met NIAM

In [FEL1989-148] werd al aandacht besteed aan de verschillende kennisniveaus die kunnen worden onderscheiden in een 'intelligent' systeem. Met name de KADS-methodiek [Breuker87] maakt een indeling naar domein-, inferentie-, taak- en strategisch niveau. Op elk van deze niveaus wordt op een bepaalde manier tegen 'kennis' aangekeken: het domeinniveau kan worden gekenschetst als het meest concreet, omdat hier de objecten in het probleemdomein en hun onderlinge relaties worden vastgelegd. Het strategieniveau biedt een veel abstracter kijk op het probleemdomein. (De KADS-theorie is hier zelfs zo abstract dat dit kennisniveau in praktijktoepassingen van deze methodiek meestal niet wordt gespecificeerd). De tussenliggende niveaus (inferentie en taak) bevatten als het ware de 'dynamiek' van het kennissysteem. Het inferentieniveau geeft aan welke elementaire inferenties (specificeren, abstraheren etc.) kunnen worden uitgevoerd met de objecten en relaties op het domeinniveau. Het taakniveau groepeer (en 'activeert') inferenties om zodoende een bepaalde taak (classificatie, diagnose etc.) vast te leggen.

Het lijkt geen twijfel dat de specificatie van het domeinniveau sterke overeenkomsten vertoont met de specificaties die worden opgesteld bij de ontwikkeling van database-systemen. Ook hier is het zaak een data-dictionary op te bouwen: een bestand dat 'aard en wezen' van elk gebruikt gegeven vastlegt (definitie, betekenis, fysieke implementatie en relaties met andere gegevens). Het inferentieniveau kan als meer procesgericht worden aangemerkt. Door aan te geven wat de invoer en uitvoer van de elementaire inferentiestappen (processen) is, kan een functioneel model van het probleemdomein worden gemaakt. Het taakniveau 'dynamiseert' dit model doordat met behulp van processpecificaties stap voor stap wordt beschreven welke actie(s) moet(en) worden uitgevoerd. Gezien de geringe praktische relevantie wordt het strategische niveau hier verder buiten beschouwing gelaten.

In het fase 1-rapport zijn drie ontwikkelmethoden de revue gepasseerd die elk de potentie hebben om een bijdrage te leveren aan de specificatie van kennissystemen: Structured Analysis (SA), Jackson System Development (JSD) en Nijssens Informatieanalyse

Methode (NIAM). De vraag in hoeverre deze methoden geschikt zijn voor het eerder genoemde doel kan niet volledig beantwoord worden! Tot nu toe zijn er niet veel praktijkgevallen bekend waarin één of meer van deze methoden zijn toegepast bij de bouw van intelligente systemen. In de literatuur wordt echter wel druk over dit onderwerp gespeculeerd.

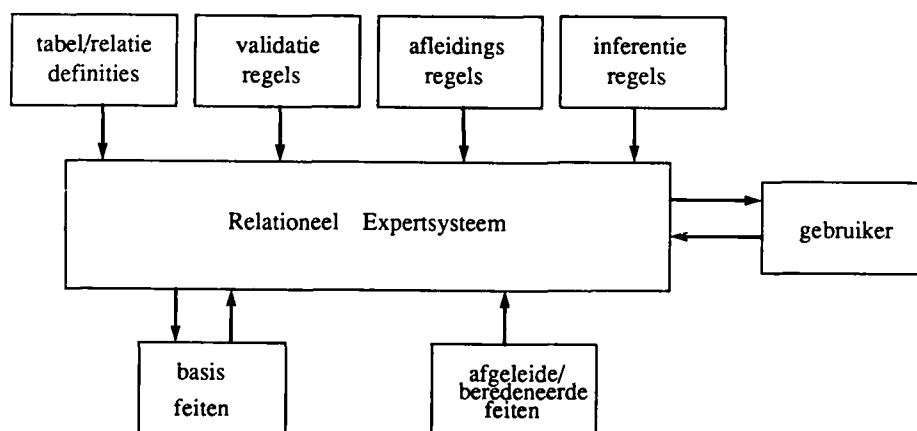
[Vogler88] stelt dat de kennisniveaus van KADS relatief eenvoudig zijn over te zetten naar SA. Het domeinniveau komt overeen met de data-dictionary die opgesteld wordt, het inferentieniveau is te vergelijken met een data-flow-diagram en het taakniveau lijkt op de definities van een processtructuur in SA. Voordeel van deze aanpak is dat de integratie met conventionele systeemontwikkeling gemakkelijker is. Nadeel blijft dat de SA-methode sterk procesgericht is. Dit bergt namelijk het gevaar in zich dat een specificatie van het domeinniveau moeilijker is te realiseren. Juist dit niveau is van essentieel belang voor een kennissysteem en is in het verleden maar al te vaak verontachtzaamd. Een toepassing kan worden gevonden in [Wagenaar88].

Er zijn (nog) geen voorbeelden bekend van kennissystemen die met JSD zijn ontwikkeld. JSD legt sterk de nadruk op modellering van objecten in de werkelijkheid (voor zover die liggen binnen de grenzen van het te ontwikkelen systeem) en de acties die ze ondergaan. Echter ook hier is de gegevensstructurering (nog) niet sterk ontwikkeld.

NIAM biedt uitstekende mogelijkheden om het domeinniveau van een kennissysteem consistent en volledig te specificeren [Bruin88]. Deze methode, die in tegenstelling tot SA en JSD gegevensgericht is, wordt o.a. gebruikt bij het ontwikkelen van het conceptueel model voor database-systemen. In [FEL1989-148] is er al op gewezen dat met NIAM tevens zogenaamde 'niet-grafische' beperkingsregels kunnen worden vastgelegd. Deze regels stellen beperkingen aan het kunnen vóórkomen van bepaalde feiten in een geautomatiseerd informatiesysteem. Met name in [Nijssen86] wordt ingegaan op de stelling dat een kennissysteem niets anders is dan een geautomatiseerd informatiesysteem dat bepaalde menselijke expertise bevat. Deze expertise (of kennis) is slechts een verzameling van aan elkaar gerelateerde feiten. Feiten kunnen door de gebruiker zijn ingevoerd (naar aanleiding van een vraag of bij de creatie van de database). Het is ook mogelijk dat ze afgeleid zijn van andere feiten die al in de database aanwezig zijn, met behulp van afleidingsregels. Tenslotte kunnen ze het resultaat zijn van een

redeneerproces: door inferentieregels toe te passen op reeds aanwezige feiten kunnen nieuwe feiten worden beredeneerd.

De beperkingsregels die kunnen voorkomen in de gedaante van afleidingsregels of inferentieregels zijn van groot belang. Door een methode te kiezen die niet alleen op het domeinniveau goede ondersteuning biedt, maar ook op abstractere kennisniveaus (middels beperkingsregels), kan de ervaring die is opgedaan in de database-wereld met het integriteitsprobleem worden getransponeerd naar de knowledgebase-wereld. In figuur 4.3 is de structuur weergegeven van wat een relationeel expertsysteem kan worden genoemd.



Figuur 4.3: Een relationeel expertsysteem.

Er is een structuursectie die de definities van de tabellen bevat, een sectie met validatieregels (de beperkingsregels uit de informatiestructuurdiagrammen van NIAM),

een sectie met afleidingsregels (die feiten kunnen afleiden die impliciet in de database aanwezig zijn) en een sectie met inferentieregels (die feiten kunnen afleiden die niet impliciet in de database aanwezig zijn). Gebruikers 'converseren' met het systeem en werken op hun eigen database, daarnaast is er nog een 'virtuele' database waarin de afgeleide en beredeneerde feiten zijn opgeslagen.

In het resterende gedeelte van dit hoofdstuk zal NIAM in het middelpunt van de belangstelling staan. Dit neemt niet weg dat aan de toepassing van deze methode in het kader van kennissystemen enkele problemen kleven. Hiervoor werd gesteld dat NIAM niet alleen kan worden toegepast op het domeinniveau, maar ook op het inferentie- en taakniveau. Echter, NIAM-specificaties op deze laatste twee niveaus bestaan uit (ongeordeerde) niet-grafische beperkingsregels. De methode biedt voor dit gedeelte geen faciliteiten om te voorkomen dat regels tegenstrijdig zijn, elkaar substitueren, circulair zijn of redundant. Het is de verantwoordelijkheid van de ontwerper om hiertegen te waken. Uitbreidingen van NIAM die zorgdragen voor een even goede handhaving van consistentie en volledigheid op het inferentie- en taakniveau als op het domeinniveau, zijn zeer welkom. Een ander nadeel is dat temporele aspecten in NIAM tot nu toe nog weinig aandacht hebben gekregen. SA, in het bijzonder de real-time uitbreidingen, en JSD bieden wat dit aspect aangaat meer.

Desondanks is hier voor NIAM gekozen, met de aantekening dat "(v)anwege de beperking van het informatiemodel tot klassen (typen) in plaats van individuele elementen en tot typering op slechts één niveau van abstractie, [...] de toepasbaarheid van NIAM voor de specificatie van op kennis gebaseerde systemen, beperkt [is] tot [...] systemen [...] die 'klassieke' gegevens en kennis combineren" [Wintraecken88]. Het is dus de vraag of NIAM zich leent voor systemen die bijvoorbeeld zintuiglijke functies moeten uitvoeren (denk aan robots). Voor omvangrijke C3I-systemen, uitgerust met intelligente modules, lijkt NIAM een methode die het overwegen méér dan waard is.

4.3.2 Beperkingsregels in NIAM

In het rapport fase 1 is al ingegaan op NIAM [FEL1989-148]. Kernpunt is dat deze methode uitgaat van een beschrijving van de werkelijkheid met behulp van natuurlijke taal. In de ontwikkeling van kennissystemen wordt enerzijds gebruik gemaakt van

kennis/informatie die schriftelijk is vastgelegd (handboeken etc.) en meer op ervaring gebaseerde kennis die middels interviews met een expert worden afgeleid. De bedoeling van kennisverwerving is een structuur aan te brengen in deze rijstebrijberg; er moet een conceptueel model van het kennisdomein worden ontworpen.

De werkelijkheid bestaat uit een verzameling objecten die in een bepaald kennisdomein relevant zijn. Deze stelling, gebaseerd op [Wintraecken85], impliceert dat informatie of overdraagbare kennis niets anders is dan het doen van een uitspraak of bewering over objecten in deze werkelijkheid. Een feit is een elementaire bewering die een verband legt tussen objecten in de werkelijkheid. Kenmerk van een elementaire bewering is dat deze slechts één enkel feit bevat. Elke niet-elementaire bewering is te splitsen in elementaire beweringen.

De objecten in een elementaire bewering kunnen worden verdeeld in lexicale en niet-lexicale. Lexicale objecten zijn op enigerlei wijze te representeren d.m.v. letters, cijfers en/of andere symbolen en zijn op te vatten als namen of verwijzingen naar andere objecten. Voor een niet-lexicaal object geldt dat het niet kan worden gerepresenteerd. De elementaire bewering: "Klaassen is de achternaam van een medewerker" bevat als lexicaal object de naam "Klaassen" en als niet-lexicaal object "medewerker", waarnaar wordt verwezen d.m.v. de naam.

NIAM legt de werkelijkheid vast met behulp van binaire feiten, d.w.z. feiten waarin steeds twee objecten uit de werkelijkheid een rol spelen. Er wordt een onderscheid gemaakt tussen twee soorten binaire feiten: bruggen en ideeën. Een brug is een feit dat betrekking heeft op een lexicaal en een niet-lexicaal object; het vormt als het ware een brug tussen twee 'werelden'. Een idee is een feit dat betrekking heeft op twee niet-lexicale objecten; er is gekozen voor de naam idee, omdat er over niet-lexicale objecten wordt 'gesproken' los van de lexicale objecten waardoor ze nader worden aangeduid. Het voorbeeld uit de laatste alinea is een brug.

Er kan ook worden gesproken over typen: objecttypen (lexicale en niet-lexicale) en feittypen (idee- en brugtypen). Door deze abstractie aan te brengen is het mogelijk om te verwijzen naar een verzameling of klasse van gelijksoortige elementen. De bewering: "Er

zijn achternamen" is een voorbeeld van een specificatie van een lexicaal objecttype. Met de bewering: "Er zijn medewerkers" wordt een niet-lexicaal objecttype gespecificeerd.

In NIAM gaat men ervan uit dat kennis d.m.v. feiten en regels kan worden beschreven. Feiten zijn op te vatten als kennis over een bepaalde werkelijkheid, die kan veranderen in de tijd. Regels zijn elementaire beweringen die geen feiten zijn; ze leggen vast welke feiten deel kunnen uitmaken van de overdraagbare kennis en welke niet. Alle feiten zijn opgenomen in de knowledgebase, terwijl de grammatica de beschrijving bevat van alle regels.

Een belangrijk type regels zijn de beperkingsregels. Dit zijn regels die geen specificatie van objecttypen of feittypen zijn; ze beperken de feiten die toegestaan zijn op grond van de definities van objecttypen en feittypen. De beperkingsregels zijn weer te onderscheiden in statische en overgangsregels. De statische beperkingsregels leggen vast welke feiten zich op elk moment in de knowledgebase mogen bevinden; ze beschrijven toestanden. De overgangsregels leggen vast welke veranderingen zich in de knowledgebase mogen voordoen; ze beschrijven toestandsovergangen.

In het algemeen verloopt een informatie/kennisanalyse m.b.v. NIAM langs de volgende lijnen. Om te beginnen moet de werkelijkheid in woorden beschreven worden. Daarna moet deze beschrijving uitgerafeld worden tot elementaire beweringen. Vervolgens worden de elementaire beweringen onderscheiden in feiten en regels. Deze laatste worden tenslotte formeel beschreven.

Er is in NIAM een onderscheid te maken tussen grafische en niet-grafische beperkingsregels, de eersten zijn weer te geven in een informatiestructuurdiagram (ISD) d.m.v. bepaalde symbolen. De laatste moeten d.m.v. tekst aan het ISD toegevoegd worden.

Een grafische beperkingsregel is de uniciteitsregel die aangeeft of er een 1:1, 1:M of N:M relatie tussen objecten bestaat. Verder is er de totaliteitsregel die bijvoorbeeld aangeeft dat alle objecten van een bepaald objecttype in een rol deelnemen (bijvoorbeeld "Alle afdelingen hebben medewerkers"). Deze beperkingsregels hebben steeds betrekking op een enkel feittype.

Er zijn ook grafische beperkingsregels die betrekking hebben op meerdere feittypen. Dit zijn de gelijkheids-, deelverzamelings- en uitsluitingsregels. Gelijkheidsregels geven aan dat de populaties van twee rollen van eenzelfde objecttype (in verschillende feittypen) gelijk zijn. Bijvoorbeeld: "Alle medewerkers hebben een bepaalde functie en zitten elk in een bepaalde salarisgroep". De gelijkheidsregel geeft aan dat de rollen waarin een medewerker participeert aan elkaar gelijk moeten zijn. Een deelverzamelingsregel geeft aan dat een rol van een objecttype een deelverzameling is van een andere rol van hetzelfde objecttype. Een uitsluitingsregel geeft aan dat bepaalde rollen van eenzelfde feittype elkaar uitsluiten (bijvoorbeeld: "Een medewerker is niet zijn eigen chef").

Uitsluitings- en totaliteitsregels gelden ook met betrekking tot subtypen. Medewerkers kunnen vaak worden onderscheiden in afdelingsmedewerkers en afdelingschefs. Deze subtype-relatie geeft aan dat 'gewone medewerkers' en 'chefs' zijn die samen de verzameling 'medewerkers' vormen. Tevens wordt aangegeven dat iemand of een 'medewerker' of een 'chef' is.

De niet-grafische beperkingsregels kunnen niet in een ISD worden opgenomen en worden daarom apart in tekst beschreven. Net als bij grafische beperkingsregels hebben de niet-grafische betrekking op de populaties van rollen in feittypen. Ze kunnen worden onderscheiden in waarderegels, kardinaliteitsregels, subtypebepalende regels, afleidbaarheidsregels, overgangsregels en inferentieregels. Deze laatsten worden in paragraaf 4.5 verder aan de orde gesteld.

Een waarderegel geeft aan welke lexicaal objecten in de populaties van een rol van een lexicaal objecttype mogen voorkomen. Bijvoorbeeld: "Voor elk geldbedrag g geldt dat als g het salaris van een medewerker is dan is $g < 150000$ ". Een kardinaliteitsregel beperkt het aantal keren dat een object van een bepaald objecttype in de populatie van een rol van dit objecttype mag voorkomen. Bijvoorbeeld: "Voor iedere afdeling a geldt dat het aantal afdelingsmedewerkers < 15 ". Een subtypebepalende regel schrijft voor wanneer een object van een bepaald objecttype tot een subtype hiervan behoort. Bijvoorbeeld: "Voor iedere afdeling a geldt dat a is een stafafdeling dan en alleen dan als a geen lijnafdeling is". Een afleidingsregel geeft aan of de populatie van een bepaald (afleidbaar) feittype aan de hand van een of meer beperkingsregels kan worden afgeleid uit de populaties van een

of meer andere feittypen. Bijvoorbeeld: "Voor iedere afdeling a geldt dat het projectbudget van a gelijk is aan de som van de geldbedragen die het budget zijn van de projecten die aan a zijn toegewezen". Een overgangsregel schrijft voor welke veranderingen binnen de populaties van een of meer rollen zijn toegestaan. Bijvoorbeeld: "Voor iedere medewerker m geldt dat het nieuwe salaris van m \geq oude salaris van m".

De hierboven gegeven voorbeelden tonen aan dat NIAM een zeer volledige specificatie kan opleveren van een bepaald domein. Dit gerelateerd aan de opmerkingen over de equivalentie tussen databases en knowledgebases opent de mogelijkheid om vollediger en consistentere specificaties te produceren bij de ontwikkeling van kennisystemen.

4.4 Beperkingsregels in database-management-systemen

Op het gebied van databases wordt het begrip integriteit in verschillende betekenissen gebruikt. In de eerste plaats geeft het aan dat een database correcte gegevens bevat, d.w.z. het moet worden uitgesloten dat het aanbrengen van veranderingen (updates) incorrecte gegevens tot gevolg heeft. Wanneer in een gegevenselement de datum is opgeslagen volgens de structuur jaar/maand/dag en een gebruiker voert deze gegevens in een andere volgorde in, dan mag dit door het systeem niet worden geaccepteerd. Een tweede betekenis van integriteit is 'consistentie', d.w.z. dat de waarden van twee of meer gegevenselementen met elkaar in overeenstemming moeten zijn. Het afdelingsnummer in een personeelsrecord moet bijvoorbeeld overeenkomen met een afdelingsnummer in een afdelingsrecord. Door een integriteits-subsysteem op te nemen binnen het database-management-systeem, kunnen bepaalde vormen van integriteit automatisch gewaarborgd worden. Dit systeem ziet erop toe dat transacties op de database correct verlopen; in het geval dat een integriteits-regel wordt geschonden zal het subsysteem een passende actie ondernemen, bijvoorbeeld het niet uitvoeren van de transactie. Hier moet echter wel opgemerkt worden dat er op dit gebied nog veel onderzoek wordt uitgevoerd: de problemen rond integriteit zijn nog niet volledig opgelost!

De integriteits-regels in een database-management-systeem (DBMS) kunnen worden verdeeld in twee soorten: domein- en relatie-integriteitsregels. Attributen in een relatie kunnen waarden aannemen die een deelverzameling vormen van een bepaald domein. Er zijn primaire domeinen (zoals 'character', 'integer' en 'floating-point') en afgeleide

domeinen (zoals 'medewerkersnaam' gedefinieerd als een string van 15 'characters'). Een domein-integriteits-regel is op te vatten als de definitie van een bepaald domein. Ter illustratie enkele voorbeelden. De structuur is steeds als volgt: op de eerste regel wordt het domein gedeclareerd, de tweede regel bevat een beperkingsregel (zie 3) of legt vast wat er moet gebeuren als de integriteitsregel wordt geschonden:

-
- ```
1) DCL MEDNR PRIMARY DOMAIN INTEGER(7)
 ELSE
 DO;
 melding("MEDNR domeinregel geschonden");
 REJECT;
 END;

2) DCL MEDNAAM DOMAIN CHARACTER(15);

3) DCL SALARIS DOMAIN FLOAT(5,1)
 SALARIS > 0 AND SALARIS < 5000;
```
- 

Figuur 4.4: Integriteitsregels.

Het is mogelijk om andere primaire domeinen in een DBMS te ondersteunen, een voorbeeld is het type 'boolean'. Daarnaast kunnen de afgeleiden domeinen ook bestaan uit een combinatie van andere domeinen; bijvoorbeeld het domein 'datum' als een combinatie van de domeinen 'dag', 'maand' en 'jaar'. De definitie van domeinen kan zodanig zijn dat er een bepaalde ordening in is aan te geven. Dit kan zowel een 'lexicografische' als een 'numerieke' ordening betreffen. Het maken van vergelijkingen tussen verschillend gedefinieerde domeinen is normaliter niet toegestaan. In de meeste gevallen is dit eenvoudig te begrijpen (bijvoorbeeld het vergelijken van een gewicht met een naam). Echter, in sommige gevallen kan het erg goed van pas komen. Een voorbeeld hiervan is de vergelijking van twee afstanden, de een in mijlen en de ander in kilometers.

Door aan de definitie van 'afstand' een conversieregel toe te voegen is dit probleem elegant op te lossen. Met behulp van de domeinen is het nu mogelijk om een relatie te definiëren:

---

```
DCL MEDEWERKER RELATION
 ATTRIBUTES (MEDNR DOMAIN (MEDNR),
 MEDNAAM DOMAIN (MEDNAAM),
 SALARIS DOMAIN (SALARIS))
 PRIMARY KEY (MEDNR)
```

---

Figuur 4.5: Definitie van een relatie.

Naast de domein-integriteitsregels zijn er ook relatie-integriteitsregels die statische en dynamische beperkingen aan de relaties in de database opleggen. Onder statische beperkingen kunnen we verstaan regels die vastleggen dat alle velden van een bepaald attribuut in een relatie groter dan nul dienen te zijn. In een medewerkersbestand is de waarde van het salaris-attribuut altijd groter dan 0. Evenals bij de domein-integriteitsregels is de structuur van de relatie-integriteitsregels: trigger-conditie, beperkingsregel, actie bij schending beperkingsregel.

Statische relatie-integriteitsregels zeggen iets over de database op een bepaald moment. Dit kan zijn op elk willekeurig moment, maar het kan ook nader zijn gespecificeerd. Voorbeelden hiervan zijn de regels met in de trigger-conditie de keywords AFTER/BEFORE UPDATE, INSERT of DELETE.

Met een dynamische relatie-integriteitsregel wordt een toestandsovergang van een database vastgelegd. Een voorbeeld hiervan is dat het gemiddelde salaris van alle medewerkers in een bestand voor een UPDATE lager is dan het gemiddelde na de UPDATE. Hierbij is ervan uitgegaan dat bij een UPDATE het salaris altijd hoger wordt.

In een dergelijke integriteitsregel wordt een vergelijking gemaakt tussen de oude en de nieuwe toestand.

Een ander aspect van relatie-integriteitsregels is dat ze betrekking kunnen hebben op zowel verzamelingen van records (tupels) als enkele records. Bij een database COMMIT worden beperkingen opgelegd aan de wijze waarop de nieuwe toestand van de oude verschilt. Een schending van een beperkingsregel met als trigger-conditie WHEN COMMITTING heeft een zogenaamde ROLLBACK tot gevolg; het terugdraaien van alle transacties tot een bepaald tijdstip in het verleden waarop de database nog consistent was. Door een beperkingsregel pas uit te voeren op het moment dat de database geCOMMIT wordt, is een voorbeeld van een 'uitgestelde' werking. Als de regel direct wordt uitgevoerd na een verandering in de database, is er sprake van een directe werking.

De relatie-integriteitsregels kunnen vanuit verschillende gezichtspunten worden ingedeeld. Tot nu toe is gesproken over statische en dynamische, record en set, directe en uitgestelde relatie-integriteitsregels.

#### 4.5 Van database naar knowledgebase

Relationele-database-management-systemen (RDBMS'en) zijn voornamelijk voor administratieve toepassingen gebruikt. Echter, concepten als data-onafhankelijkheid, data-integriteit, gecontroleerde redundantie, veiligheid en privacy zijn ook van belang voor knowledgebase-management-systemen (KBMS'en). Andere redenen om vast te houden aan het relationele model zijn de conceptuele eenvoud en de bekendheid bij ontwikkelaars van systemen. Door een RDBMS te gebruiken kunnen update-anomalieën worden voorkomen, evenals redundante opslag. Verder biedt het een flexibel groeipad wanneer operationele concepten veranderen en wijzigingen moeten worden aangebracht in de structuur van de gegevens.

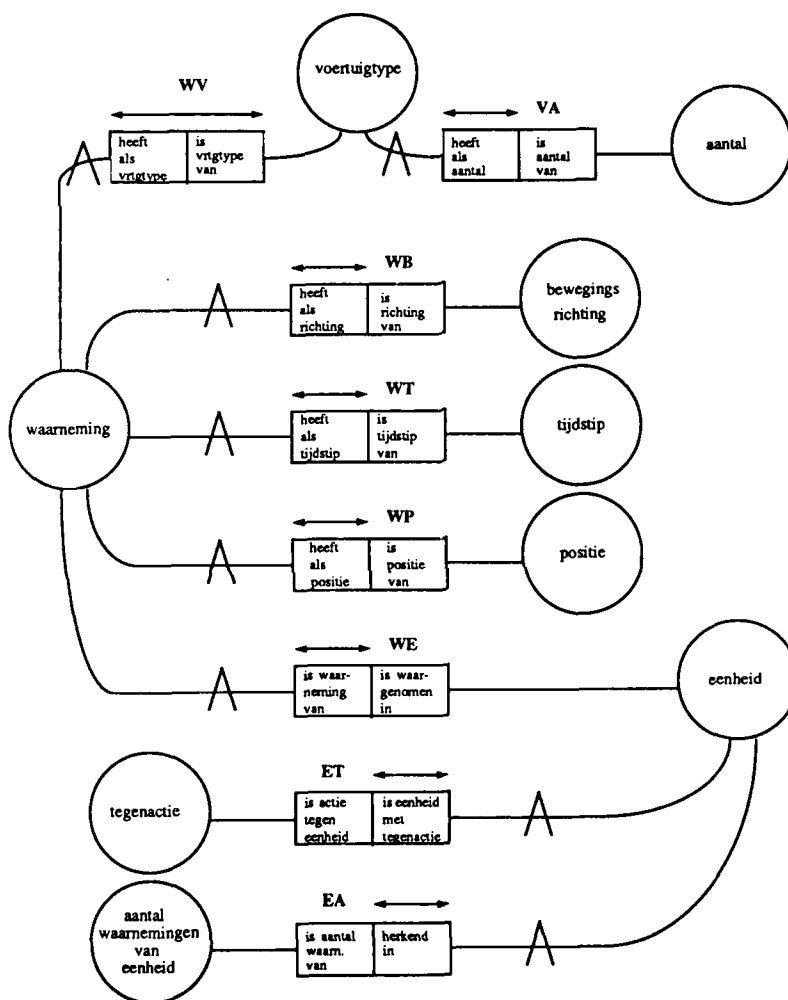
Door een knowledgebase te beschouwen als een speciale database kunnen verschillende faciliteiten die in DBMS'en gebruikt zijn ook worden toegepast in KBMS'en. Voorbeelden zijn recovery (het herstel van de database na een calamiteit, fout of spanningsuitval), concurrency (het gelijktijdig gebruik maken van de database door verschillende personen), security (de beveiliging van de database tegen onbevoegd

gebruik) en integrity (het bewaken van de consistentie van de database). Juist op dit laatste punt is er een directe relatie te leggen met de analyse- en ontwerpmethoden voor databases. Globaal komt het erop neer dat er een conceptueel model van het kennisdomein moet worden opgesteld. Het conceptuele model ligt in tussen het interne model van de database (de wijze waarop de relaties fysiek in de database zijn opgenomen) en het externe model (de wijze waarop de gebruiker tegen het systeem aankijkt). Het conceptuele model moet een volledige en consistente weergave van het kennisdomein zijn, waarin net als bij een database-toepassing een onderscheid wordt gemaakt tussen een kennisschema (de definities van alle voorkomende feiten en relaties) en de werkelijke knowledgebase. Dit onderscheid kan ook opgevat worden als een duidelijke scheiding tussen typen en instanties. Het voordeel is dat er nu op een geabstraheerde manier over de knowledgebase kan worden gesproken. Consistentie en volledigheid zijn te bewaken d.m.v. beperkingsregels die aan de werkelijke knowledgebase worden opgelegd. Een conceptueel model dient te worden opgesteld met een methode die het onderscheid tussen feiten en hun definitie expliciet vastlegt. In dit rapport is dan ook gekozen voor NIAM.

In het voorgaande is gesteld dat een knowledgebase een verzameling feiten is die met behulp van 'intelligente' regels kan worden gemanipuleerd. Deze regels zijn zelf ook weer als objecten in de knowledgebase opgeslagen. [Nijssen85] geeft een presentatie van de equivalentie van databases en knowledgebases.

Met een voorbeeld kan een en ander duidelijker worden gemaakt. Het heeft betrekking op een militair probleem, t.w. het herkennen van eenheden op basis van voertuig-waarnemingen die in het veld worden gedaan. Voor de goede orde zij hier opgemerkt dat de in deze paragraaf gebruikte voorbeelden gefingeerd zijn.

Een inlichtingenofficier krijgt rapportages van verschillende waarnemingen voor zich. Van elke waarneming zijn een aantal attributen bekend: geografische positie, tijdstip, voertuig/aantal combinaties en bewegingsrichting van de voertuigen. Op grond van deze gegevens kan de inlichtingenofficier eenheden herkennen. Een voorbeeld van een waarneming zou kunnen zijn: "positie FU3004, tijdstip 1700, 3xT-72, 4xBRDM, 2xBMP, NO-ZW", wat betekent dat een colonne van 3 T-72 tanks, 4 BRDM gepantserde wielvoertuigen en 2 gepantserde rupsvoertuigen om vijf uur 's middags vanuit het



Figuur 4.6: ISD van het slagorde-voorbeeld.

noordoosten naar het zuidwesten langs kaartpositie FU 3004 reden. De inlichtingenofficier herkent in deze gegevens een verkenningscompagnie van een gemechaniseerd infanterieregiment. Als we ervan uitgaan dat de inlichtingenofficier een database wil bijhouden van waarnemingen, herkende eenheden en de hiertegen te ondernemen acties, dan kan dit met het ISD in figuur 4.6 worden weergegeven.

De wijze waarop de inlichtingenofficier de eenheden herkend is voor de database-toepassing niet van belang. Het gaat erom dat hij op grond van waarnemingen eenheden herkent. Het uiteindelijke resultaat wordt in de database ingevoerd. De database bevat drie tabellen waarvan hieronder de SQL-definities zijn opgenomen:

---

CREATE TABLE WAARNEMING

```
(WAARNEMINGS_ID CHAR(12) NOT NULL,
 POSITIE CHAR(10) NOT NULL,
 TIJDSTIP CHAR(6) NOT NULL,
 RICHTING CHAR(5) NOT NULL,
 EENHEIDS_ID CHAR(20)
)
```

CREATE TABLE EENHEID

```
(EENHEIDS_ID CHAR(20) NOT NULL,
 TEGENACTIE CHAR(15) NOT NULL,
 AANTAL_WAARN SMALLINT NOT NULL)
```

CREATE TABLE VOERTUIG\_AANTAL

```
(WAARNEMINGS_ID CHAR(12) NOT NULL,
 VOERTUIG_TYPE CHAR(15) NOT NULL,
 AANTAL SMALLINT NOT NULL)
```

---

Figuur 4.7.1: SQL-definities van het slagorde-voorbeeld.

---

```
CREATE UNIQUE INDEX I1
 ON WAARNEMING (WAARNEMINGS_ID)

CREATE UNIQUE INDEX I2
 ON EENHEID (EENHEIDS_ID)

CREATE UNIQUE INDEX I3
 ON VOERTUIG_AANTAL (WAARNEMINGS_ID,VOERTUIG_TYPE)

CREATE REFERENTIAL INTEGRITY RULE R1
 WAARNEMING.EENHEIDS_ID
 IN
 EENHEID.EENHEIDS_ID

CREATE REFERENTIAL INTEGRITY RULE R2
 VOERTUIG_AANTAL.WAARNEMINGS_ID
 IN
 WAARNEMING.WAARNEMINGS_ID
```

---

Figuur 4.7.2: SQL-definities van het slagorde-voorbeeld.

Bij de totstandkoming van de database zijn globaal genomen drie personen betrokken. Ten eerste is er in het veld een waarnemer aanwezig die wat hij ziet doorgeeft aan de inlichtingenofficier: voor het gemak wordt hier over één (virtuele) waarnemer gesproken, in werkelijkheid zullen het er meer zijn. Ten tweede de inlichtingenofficier die op grond van waarnemingen eenheden herkent en acties voorstelt om deze (vijandelijke) eenheden aan te vatten. Ten slotte beschikt de inlichtingenofficier over de diensten van een sergeant die bijhoudt hoe vaak een bepaalde eenheid is waargenomen (meer specifiek, op welke posities dezelfde eenheid is waargenomen).

Op dit punt aangekomen vraagt Nijssen zich af of er bepaalde vormen van expertise in het database-systeem kunnen worden opgenomen. De activiteiten van de waarnemer zijn dusdanig dat ze direct (misschien beter gezegd door tussenkomst van een berichtenklerk) in de database kunnen worden opgenomen. De werkzaamheden van de sergeant kunnen door het systeem worden overgenomen: met een afleidingsregel stelt het systeem zelf vast hoe vaak een eenheid is waargenomen. Hieronder staan twee SQL-statements waarin een tabel wordt gegenereerd voor het aantal waarnemingen van een eenheid, en een UPDATE-statement waarmee deze relatie consistent met de rest van de database kan worden gehouden. Dit laatste is een relatie-integriteitsregel die wordt getriggerd op het moment dat er een verandering plaatsvindt in de waarnemingsrelatie:

---

```
INSERT INTO EW
 SELECT IS_WAARGENOMEN_IN, COUNT(*)
 FROM WE
 GROUP BY IS_WAARGENOMEN_IN
UPDATE EENHEID
 SET AANTAL_WAARN= (SELECT COUNT(*)
 FROM WAARNEMING
 WHERE WAARNEMING.EENHEIDS_ID=
 EENHEID.EENHEIDS_ID)
```

---

Figuur 4.8: Voorbeeld van een afleidingsregel.

Nu deze afleidingsregel in het systeem is opgenomen is er nog niet echt sprake van een 'intelligent' systeem, maar dit wordt anders als het redeneerproces van de inlichtingenofficier aan het systeem wordt toegevoegd. Door deze persoon interviews af te nemen kan worden vastgesteld hoe dit redeneerproces is gestructureerd. Hieronder is een inferentieregel opgenomen, samen met het SQL-equivalent. Deze inferentieregel kan niet op een grafische wijze worden gerepresenteerd in ISD's en zal als 'niet-grafische beperkingsregels' aan de ISD's worden toegevoegd:



---

INFERENTIEREGEL :

```
IF HEEFT_ALS_VRTGTYPE = BMP OR BTR OR 60-PB
 AND (AANTAL > 8 AND AANTAL <13)
 THEN EENHEID_ID = MECHINFCIE
```

## SQL-STATEMENT :

```
UPDATE WAARNEMING
SET EENHEIDS_ID = MECHINFCIE
 WHERE HEEFT_ALS_VRTGTYPE = BMP
 OR HEEFT_ALS_VRTGTYPE = BTR
 OR HEEFT_ALS_VRTGTYPE = 60-PB
 AND AANTAL > 8
 AND AANTAL < 13
```

---

Figuur 4.9: Voorbeeld van een inferentieregel.

Inmiddels zijn geïntroduceerd een uitleg van de verschillende soorten integriteit in databases en knowledgebases; de manier waarop met NIAM een conceptueel kennismodel kan worden vastgelegd; een illustratie van het nut van deze aanpak met een klein voorbeeld. Er werd steeds van uit gegaan dat het systeem waarmee gewerkt wordt een traditioneel RDBMS-pakket, zoals ORACLE, is. Bij de inferentie- en afleidingsregels is SQL-code gebruikt die als trigger in het systeem ligt opgeslagen. Nieuwe ontwikkelingen op het front van RDBMS'en hebben geleid tot architecturen waar het voorgaande nog beter in kan worden vastgelegd. Eén hiervan, POSTGRES, zal in hoofdstuk 5 worden besproken.

#### 4.6 Gestructureerde kennismodellen in slecht gestructureerde domeinen

Bij een 'traditionele' database-aanpak is een zekere mate van structurering van de kennisbank - of tenminste het feitengedeelte daarvan - al in de ontwerpfase gedefinieerd. Soms zal dat voldoende zijn, maar in andere gevallen schiet een dergelijke mate van structurering tekort. In een deel van deze gevallen kan een blackboard aanpak wel voor de benodigde structurering zorgen. Ook een 'moderne' database-aanpak zoals die gerealiseerd is in het in Hoofdstuk 5 te bespreken POSTGRES-systeem kan wat dit betreft meer dan traditionele database-systemen als INGRES en ORACLE.

Het bieden van de mogelijkheid om kennisbank en kennismodel te structureren is echter nog nauwelijks ondersteuning van modulaair ontwerp te noemen. Dat er structuur aangebracht kan worden is fijn, maar de vraag is vaak *hoe* dat moet gebeuren. Bij kennissysteem-ontwikkeling zijn kennisingenieur en domeinspecialist vaak niet dezelfde mensen en ook wanneer dat wel het geval is blijkt het aanbrengen van een geschikte structuur in een in eerste instantie ongestructureerd domein vaak geen sinecure.

De mate waarin het domein van een te ontwikkelen kennissysteem een (overzichtelijke) structurering bezit is dus een belangrijk criterium voor de wijze waarop de ontwikkeling ter hand genomen moet worden alsook voor de mate waarin met een kennisacquisitie-bottleneck rekening gehouden moet worden. In [Kuhn62] worden de termen paradigmatisch en pré-paradigmatisch gebruikt voor wetenschapsgebieden, die hierboven als duidelijk respectievelijk niet duidelijk gestructureerd zijn omschreven. Wiskundige en natuurkundige theorieën zijn in het algemeen goede voorbeelden van paradigmatische theorieën. Ze zijn opgebouwd rond een betrekkelijk klein aantal, scherp gedefinieerde (basis-)begrippen en bezitten een sterk hiërarchische structuur. Aan het andere eind van het spectrum staan (de meeste) sociaal-wetenschappelijke en medische theorieën, die een veel groter aantal concepten bevatten waarvan de onderlinge samenhang veelal ruimte voor interpretatie openlaat.

Paradigmatische kennis zal zich meestal gemakkelijk in een database-structuur laten vastleggen via een specificatie-methode als NIAM. Bij pré-paradigmatische kennis echter, waar juist het selecteren en definiëren van de te gebruiken concepten een probleem is zal de kennismodellering onherroepelijk een bottleneck zijn. De eis van een

overzichtelijk kennismodel maakt de zaak hier alleen nog maar erger (d.w.z. trager). In feite kan pré-paradigmatische kennis het *gemakkelijkst* worden vastgelegd in oppervlakkige produktieregels. Wanneer men echter lukraak voor deze optie kiest en het daarbij laat, levert dat - zoals in de praktijk is gebleken - een systeem op dat schromelijk tekort schiet in betrouwbaarheid, onderhoudbaarheid, uitlegfaciliteiten en graceful degradation.

Eén manier om in een dergelijke situatie te werk te gaan is de eis van een overzichtelijk kennismodel in eerste instantie te laten varen en met behulp van rapid prototyping eerst een kennisbank met conceptueel rommelige en oppervlakkige produktieregels te ontwikkelen. Aan de hand van dit eerste prototype kan dan gepoogd worden middels de selectie (of liever creatie) van een beperkt aantal begrippen (relaties) een raamwerk te fabriceren waar de kennis van het betreffende domein met enig knip- en kneed-werk ingewrongen kan worden. Deze formulering klinkt niet erg vriendelijk, maar hier is in feite sprake van theorievorming en een (niet al te grove) vervorming van de werkelijkheid is daarbij noodzakelijk om een enigszins betrouwbaar en onderhoudbaar kennissysteem op te leveren binnen een beperkt budget.

In [Bakker87] worden concepten en algoritmes beschreven die van nut kunnen zijn bij de (hiërarchische) structurering van een kennismodel in de vorm van een gerichte, gelabelde graaf. De identificatie van subgrafen die qua *structuur* overeenkomst vertonen (d.w.z. overeenkomstige relaties hebben met dezelfde punten in de rest van de graaf) wordt hierbij beschouwd als een geschikte *heuristiek* om groepen concepten (inclusief de relaties ertussen) te selecteren, die gebundeld kunnen worden in nieuwe complexere begrippen en op deze wijze een hiërarchisch gestructureerd kennismodel opleveren. Of deze heuristiek in de praktijk werkt zal nog moeten blijken. Dat het (niet meer dan) een heuristiek is zeker. Immers, structurele overeenkomst garandeert nog geen conceptuele overeenkomst, aangezien ook de oorspronkelijke kennisgraaf slechts een afspiegeling van de werkelijkheid is. Wij verwachten dat dergelijke 'abstracte' operaties op grafen wellicht een verlichting, doch geen opheffing van het kennismodelleringsprobleem voor pré-paradigmatische kennis kunnen opleveren.

## 5 KWALITEIT VAN HET PRODUKT

### 5.1 Inleiding

Zoals eerder in hoofdstuk 2 aangestipt worden de onderstaande aspecten in de kwaliteit van kennissysteem software onderscheiden:

---

#### *Kwaliteit van het produkt*

|  |                                                     |                          |                                 |
|--|-----------------------------------------------------|--------------------------|---------------------------------|
|  | bruikbaarheid                                       |                          |                                 |
|  |                                                     | betrouwbaarheid          |                                 |
|  |                                                     |                          | integriteit                     |
|  |                                                     |                          | consistentie (x)                |
|  |                                                     |                          | volledigheid (x)                |
|  |                                                     |                          | correctheid                     |
|  |                                                     |                          | robuustheid                     |
|  |                                                     |                          | 'fail-safe' c.q. 'monkey-proof' |
|  |                                                     |                          | graceful degradation            |
|  |                                                     |                          | testbaarheid                    |
|  |                                                     |                          | modulariteit (x)                |
|  |                                                     |                          | aansluiting op specificatie (x) |
|  |                                                     | efficiëntie              |                                 |
|  |                                                     |                          | computationele efficiëntie      |
|  |                                                     |                          | gebruikersvriendelijkheid       |
|  | aanpasbaarheid                                      |                          |                                 |
|  |                                                     | onderhoudbaarheid        |                                 |
|  |                                                     |                          | leesbaarheid                    |
|  |                                                     |                          | modulariteit (x)                |
|  | overdraagbaarheid                                   |                          |                                 |
|  |                                                     | apparatuur onafhankelijk |                                 |
|  | integreerbaarheid (koppeling aan 'andere' software) |                          |                                 |

---

Figuur 5.1: Kwaliteit van het produkt.

Omdat het onderwerp produktkwaliteit als geheel te veel aspecten omvat om binnen dit project volledig onderzocht te worden is een selectie gemaakt. Deze is in bovenstaand overzicht middels kruisjes aangegeven. De selectie bestaat uit integriteit, testbaarheid en onderhoudbaarheid, geconcretiseerd in respectievelijk constraints voor consistentie, volledigheid en andere vormen van integriteit alsmede modulariteit. Integriteits-constraints kunnen beschouwd worden als tegenwicht voor de gebrekkige testbaarheid en kunnen bijdragen aan goede onderhoudbaarheid. Modulariteit speelt zowel een rol in testbaarheid als onderhoudbaarheid.

Het feit dat bovengenoemde zaken geselecteerd zijn betekent uiteraard niet dat de overige onbelangrijk zijn. Ze zijn echter veelal wat moeilijker te behandelen zonder ze te koppelen aan specifieke domeinen (bijvoorbeeld computationele efficiëntie) of performance eisen (bijv. fail safe voor real time systemen), of ze zijn minder relevant voor 'het kwaliteitsprobleem' bij kennissystemen. Overdraagbaarheid is bijvoorbeeld een aspect dat bij kennissystemen niet of nauwelijks anders geconcretiseerd wordt dan bij andere software. Gebruikersvriendelijkheid (kwaliteit van het user-interface) is een aspect dat door velen gezien wordt als het allerbelangrijkste. Zaken als hypertext en user models kunnen de kwaliteit hiervan zeer ten goede komen [Conklin87, Schneider84]. Het is echter nauwelijks kennissysteem-specifiek te noemen, afgezien van de *inhoudelijke* kwaliteit van uitleg die door het systeem gegenereerd dient te worden. Voor zover hiervoor domein-onafhankelijke richtlijnen kunnen worden gegeven komen die echter neer op expliciete representatie van meta-kennis, hetgeen als een vorm van structurering/modulariteit kan worden opgevat [Wognum89]. Modulariteit lijkt een software-karakteristiek te zijn die bij bestudering van kwaliteitsbeheersing van kennissystemen in alle hoeken en gaten de kop opsteekt. In dit rapport gebeurt dit in de hoek van de testbaarheid, omdat ook het tweede aandachtspunt, integriteit, hier een uitvloeisel van is.

## 5.2 Testbaarheid van kennissystemen

### 5.2.1 Conventionele testmethodologie

Methoden om de betrouwbaarheid van software te waarborgen zijn onder te verdelen in methoden die (grotere) betrouwbaarheid 'inbakken' (bijvoorbeeld gestructureerde analyse-ontwerp- en implementatie-methoden) en methoden die betrouwbaarheid 'achteraf' controleren. De overlappende begrippen testen, evalueren, valideren en verifiëren horen bij deze laatste groep. In de loop der jaren is voor het testen van software een uitgebreid arsenaal aan methoden, paradigma's en tips ontwikkeld. Uitgangspunt van deze paragraaf is een test-methodologie zoals die in [Myers79] is beschreven. Terwijl deze methodologie bijzonder waardevol is voor het testen van 'conventionele' programmatuur als vertalers en besturingssystemen lijkt ze voor kennissystemen en (in iets mindere mate) andere data-gedomineerde systemen als databases niet adequaat [Deutsch82]. Het is de vraag waar dit precies aan te wijten is. Om hierin inzicht te verkrijgen zullen om te beginnen wat technieken van Myers' test methodologie de revue passeren. Testen 'volgens Myers' dient deels via executie van de software en deels uit het blote hoofd te geschieden.

Van de mentale technieken kunnen inspectie en walkthrough genoemd worden. Walkthrough komt neer op het aan de hand van het bronprogramma nagaan hoe het vertaalde programma zal reageren op bepaalde testcases. Zoals bij iedere vorm van testen dient de verwachte/gewenste output voor het eigenlijke testen te worden gespecificeerd. Inspectie bestaat in twee vormen: een mondelinge parafrase van de programmeur van wat het programma doet (aan de hand van het bronprogramma) en het aflopen van een checklist van veel voorkomende fouten. Checklists zijn in het algemeen taal- en applicatie-afhankelijk. De in [Myers79] gepresenteerde checklist bevat bijvoorbeeld items als 'index within bounds?' Dit heeft weinig nut wanneer men in een taal als PASCAL programmeert: Mits gebruikmakend van een redelijk goede compiler zal een eventuele 'index out of bounds'-fout automatisch worden gedetecteerd bij een poging tot compilatie.

Ook testen via executie omvat twee typen, te weten black box en white box testen. Bij het eerstgenoemde type wordt het systeem c.q. de betreffende 'module' als een black box

beschouwd: Er wordt louter gekeken naar het resultaat (de uitvoer) bij het aanbieden van bepaalde inputs. Hoe de software daarbij tot die uitvoer komt speelt bij black box testen geen rol. Omdat het bij de meeste software geen haalbare zaak is alle mogelijke invoerwaarden uit te proberen wordt gepoogd een zo efficiënt mogelijke set van testcases samen te stellen. Efficiënt betekent hier: klein en met grote kans op het optreden van fouten. De ervaring leert hierbij dat het waardegebied van de invoer opgesplitst kan worden in zogenaamde 'equivalentie-klassen', gekenmerkt door het feit dat twee testcases uit dezelfde equivalentie-klasse tot dezelfde aanleiding zullen geven (als ze al een fout opleveren). Uiteraard is het vaststellen van equivalentie-klassen bij een black box een kwestie van gokken: Men heeft alleen de specificatie om zich aan vast te houden. Toch kan op basis daarvan vaak een redelijke gok gedaan worden. Om bijvoorbeeld equivalentie-klassen te kiezen voor een module die volgens de (summiere) specificatie 'de wortel trekt uit een getal' zou men kunnen redeneren als volgt. Men kan wat betreft de functie die aan  $x$  de wortel uit  $x$  toevoegt binnen de reële getallen de deelverzamelingen  $x \mid \sqrt{x}$  niet gedefinieerd  $x \mid \sqrt{x} \geq x$  en  $x \mid \sqrt{x} < x$  onderscheiden. Daarnaast kan men het onderscheid maken tussen gehele en gebroken getallen. (Het onderscheid tussen rationaal en reëel is uiteraard niet van belang: Normaliter kunnen slechts rationale getallen worden gepresenteerd) Naast het begrip equivalentie-klasse speelt dan nog het begrip 'grenswaarde' een rol: Het is een goede heuristiek gebleken om in ieder geval testcases met waarden op de grens van een equivalentieklasse te gebruiken. In het onderhavige geval zou dit bijvoorbeeld kunnen leiden tot de set van testcases  $(-1, ?)$ ,  $(0, 0)$ ,  $(0.01, 0.1)$ ,  $(1, 1)$ ,  $(2.25, 1.5)$ ,  $(1.0E30, 1.0E15)$ . Door het heuristische karakter van het gebruik van equivalentie-klassen en grenswaarden is het uiteraard niet objectief vast te stellen wat nu precies de 'beste' set van testcases is. Bovendien is optimaliteit altijd een kwestie van afweging tussen het vergroten van de kans om fouten op te sporen en het beperken van de tijd die met testen gemoeid is.

White box testen maakt in tegenstelling tot black box testen gebruik van de structuur van het bronprogramma. De gedachte achter de door Myers gepresenteerde white box technieken is dat een ideale set van testcases letterlijk alle hoeken en gaten van een programma bestrijkt. Dit ideaal wordt het best benaderd met 'path coverage'. Bij path coverage wordt ieder mogelijk executiepad in het programma door (tenminste) een testcase bestreken. In het algemeen leidt de eis van path coverage tot onaantvaardbaar hoge kosten. (Men moet hierbij bedenken dat een pad dat twee maal een of andere

herhalingslus doorloopt een ander pad is dan eentje die dezelfde lus drie maal doorloopt.) Daarom heeft men een reeks van coverage-criteria opgesteld, die naar oplopende veeleisendheid besproken zullen worden.

- a) *Statement coverage*: Ieder statement moet bij het testen tenminste één keer worden uitgevoerd.

Voorbeeld: Om "IF c1 OR c2 THEN a1 ELSE a2" te testen met statementcoverage zijn tenminste twee testcases nodig: een waarbij a1 uitgevoerd wordt en een tweede waarbij a2 uitgevoerd wordt.

- b) *Decision coverage*: Iedere vertakking (uitkomst van een beslissing) moet bij het testen tenminste een keer doorlopen worden.

Voorbeeld: Om "IF c1 OR c2 THEN a1" te testen met decision coverage zijn tenminste twee testcases nodig: een waarbij "c1 OR c2" TRUE oplevert (en a1 dus wordt uitgevoerd) en een waarbij het FALSE oplevert. Merk op dat een testcase hier zou volstaan voor statement coverage. Decision coverage impliceert statement coverage.

- c) *Condition coverage*: Iedere 'conditie' (term in een logische expressie) moet bij het testen tenminste een keer TRUE en tenminste een keer FALSE opleveren.

Voorbeeld: Om "IF c1 OR c2 THEN a1 ELSE a2" te testen met condition coverage zijn tenminste twee testcases nodig, bijvoorbeeld een waarbij c1 TRUE en c2 FALSE oplevert en een waarbij het omgekeerde het geval is. Merk op dat in dit geval niet aan decision coverage en zelfs niet aan statement coverage voldaan is.

- d) *Decision/Condition coverage*: Er moet zowel aan decision als aan condition coverage voldaan zijn.

Voorbeeld: Om hieraan bij het voorgaande voorbeeld te voldoen volstaan twee testcases, te weten een waarbij c1 en c2 allebei tot TRUE evalueren en een waarbij ze beide FALSE opleveren. Decision/condition coverage is zonder meer een beter criterium dan



condition coverage: de test-set bestrijkt meer van het programma, maar hoeft niet groter te zijn!

- e) *Multiple condition coverage*: Bij het testen moet het tupel van 'conditions' in een 'decision' alle mogelijke waarden tenminste eenmaal doorlopen.

Voorbeeld: Om "IF c1 OR c2 THEN a1 ELSE a2" te testen met multiple condition coverage zijn hoe dan ook 2 tot de macht 2 = 4 testcases nodig, een waarbij beide condities TRUE opleveren, een die beide FALSE doet zijn en twee waarbij de een TRUE en de ander FALSE oplevert. Multiple condition coverage impliceert decision/conditioncoverage maar vereist dan ook in het algemeen meer (tot veel meer) testinspanning. Multiple condition coverage impliceert nog lang geen path coverage, zelfs niet bij een programma zonder herhalingslussen (een beslisboom dus). Immers, per *decision* worden alle mogelijke combinaties van conditie-uitkomsten getest. Path coverage zou impliceren dat alle mogelijke combinaties van *decision*-uitkomsten (per *programma*) worden getest.

Het is redelijk te veronderstellen dat niet geneste decisions onafhankelijk van elkaar bekeken kunnen worden. Dit gaat echter niet op voor geneste decisions, in het bijzonder wanneer multiple condition coverage geëist wordt. In [Myers79] worden hierover geen duidelijke uitspraken gedaan. Wel wordt herhaaldelijk aangegeven dat multiple condition coverage een noodzakelijke voorwaarde is voor verantwoord testen. Wanneer men ervan uitgaat dat geneste decisions als één decision beschouwd worden lijkt een grootste aantal condities in de buurt van de 20 bij omvangrijke applicaties niet onmogelijk. Dit zou neerkomen op een aantal testcases in de orde van grootte van een miljoen. men kan dus gevoegelijk aannemen dat Myers ervan uitgaat dat condities in geneste decisions elkaar niet of niet altijd beïnvloeden.

#### 5.2.2 Toepasbaarheid van conventionele testmethodologie op kennissysteemsoftware.

Afgaande op uitlatingen in velerlei publikaties over de 'intricate flow of control' die verantwoord testen van kennissystemen zou belemmeren en 'complicaties bij het testen van data-gedomineerde systemen' wordt de conclusie getrokken dat de specifieke

problemen bij het testen van kennissystemen in het *declaratieve* gedeelte (de kennisbank(en)) zit en niet zozeer in de inferentie-machine (shell) [Llinas87, Myers79, Deutsch82]. Men voert in het algemeen als voordeel van expliciete (declaratieve) kennisrepresentatie boven impliciete (procedurele) aan dat shell en kennisbank afzonderlijk getest kunnen worden. Bij het bespreken van de toepasbaarheid van de eerdergenoemde white- en black-box testmethoden wordt aangenomen dat er geen bugs (meer) in de shell zitten.

Het bij black box testen toonaangevende concept equivalentie-klasse kan slechts van nut zijn bij een specificatie die ofwel gedetailleerd is of door verschillende personen (bijna) altijd op dezelfde wijze zal worden uitgewerkt. Voor black-box testen is bovendien gewenst dat de specificaties van de te testen boxes niet te groot zijn, in de zin dat het aantal benodigde testcases niet te groot is. In [Doyle85] wordt opgemerkt dat "bij de AI-aanpak (...) de geïnterpreteerde specificatie als een goedkoop, maar volledig prototype fungeert". Hier wordt de kennisbank dus beschouwd als een (deel-)specificatie die samen met de (conventionele) specificatie van de shell een specificatie van het kennissysteem is. Dit gaat wellicht op als de kennisbank relatief klein en overzichtelijk is, maar als dat niet zo is, dan lijkt het ongepast om de kennisbank als specificatie aan te merken. Een eis die aan iedere specificatie gesteld moet worden is namelijk dat de validiteit ervan zonder problemen kan worden gecontroleerd. Als de kennisbank onoverzichtelijk is, dan zal dit niet het geval zijn. De grote vraag is nu natuurlijk of er in zo'n geval wel een specificatie van de kennisbank geformuleerd kan worden waarmee de gesignaleerde test- en onderhouds-problemen kunnen worden verlicht of opgeheven. Na lezing van het voorgaande hoofdstuk zal het de lezer niet verbazen dat ons antwoord op deze vraag een voorzichtig (wat betreft het opheffen) tot volmondig (wat betreft het verlichten) 'ja' is. Voor het *regel*-gedeelte van een kennissysteem lijkt de eerder geciteerde uitspraak van Doyle inderdaad op te gaan. In een dergelijke situatie vallen black-box en white-box methoden dus samen. De enige manier om de overzichtelijkheid van produkt *alias* specificatie te vergroten is dan een herformulering van de regels, bijvoorbeeld gebruikmakend van andere domein concepten of van een totaal andere, meer modulariteit opleverende aanpak. Het behoeft geen betoog dat dit in het algemeen geen eenvoudige opgave is.

Het gebruik van *grenswaarden* in testcases zal in het algemeen minder vruchten afwerpen bij kennissystemen, eenvoudig weg omdat variabelen vaker symbolisch en dus discreet of zelfs binair zijn. *Illegale* waarden kunnen echter wel een zeer nuttige rol spelen, met name waar het opstellen van testcases gecompliceerd wordt doordat de opsteller niet zelf bij machte is de gewenste (juiste) output bij inputs te geven. Beschouw bijvoorbeeld een beïnvloedingsgraaf van fysiologische variabelen met aan de uiteinden waarneembare verschijnselen (symptomen of ziektebeelden). Men moet in het algemeen een expert (arts) zijn om een (minimale) gemeenschappelijke oorzaak van de symptomen hypertensie (te hoge bloeddruk) en haematurie (bloed in de urine) te kunnen ophoesten. Een mespuntje common sense volstaat echter om te beredeneren dat een combinatie van coma en hyperventilatie illegaal is. De testcase (coma, hyperventilatie, ??) kan dus zonder hulp van de expert worden opgesteld.

(Deze testcase is ontleend aan de praktijk; hij toonde onmiddellijk een essentiële ontwerpfout in het betreffende algoritme aan.) Wanneer een kennissysteem aan multiple condition coverage (over geneste decisions) onderworpen wordt zal dit al gauw aanleiding geven tot een onaanvaardbaar hoog aantal testcases. Het feit dat een expert meestal zal moeten assisteren bij het koppelen van gewenste outputs aan de inputs van de testcases maakt het probleem nog een graadje erger. Immers, de beschikbaarheid van experts is een bekende flessenhals bij de ontwikkeling van kennissystemen.

Vasthouden aan beperkte multiple condition coverage (d.w.z. binnen een decision) is echter, zeker in het geval van een beslisboom, in vele gevallen twijfelachtig. Een - liefst hiërarchische - opdeling van de kennisbank in kleine module is daarom zeer wenselijk. Multiple condition coverage kan dan binnen iedere module worden geëist. Als de module niet te groot zijn en de betiteling module waar kunnen maken, m.a.w. conceptuele eenheden vormen zonder al te veel wisselwerking, dan is een dergelijke aanpak ongetwijfeld optimaal. Ziedaar het vitale belang van structureren en modulariteit voor de testbaarheid. Wellicht ten overvloede moet hierbij wel worden opgemerkt dat het bovenstaande 'eisenpakket' zonder enige consideratie met de *haalbaarheid* ervan is neergeschreven. Het is een schets van de structuur die voor testen en onderhoud *ideaal* zou zijn. In de praktijk zal een en ander (bijvoorbeeld de hiërarchische structureren) te star zijn om als effectieve richtlijn te kunnen fungeren.

### 5.3 Structurering en modulariteit

#### 5.3.1 Modulariteit in conventionele software

In traditionele zin is een software ontwerp goed modulair gestructureerd als de modules een bepaalde mate van onafhankelijkheid van elkaar bezitten en iedere module functioneel gezien coherent is, d.w.z. dat de akties binnen de module aan elkaar gerelateerd zijn. Het eerste aspect wordt met 'koppeling' aangeduid, het tweede met 'cohesie'. Deze concepten zijn niet onafhankelijk van elkaar. Naarmate modules een sterkere cohesie bezitten volstaat in het algemeen een zwakkere koppeling om het gehele systeem naar behoren te laten functioneren.

Koppeling is de mate van afhankelijkheid tussen twee modules. Deze vrij abstracte definitie kan geoperationaliseerd worden als 'de kans dat bij het coderen, debuggen of modificeren van een module rekening gehouden moet worden met de inhoud van de andere module'. De koppeling tussen modules omvat de volgende vier aspecten:

a) *Aard van de verbinding tussen modules*

Een verbinding via het zenden van boodschappen en reacties zoals dat bij object-georiënteerd programmeren gebeurt is van een geheel andere aard dan het gemeenschappelijk gebruik maken van een globaal data gebied (*common environment-coupling*). Min of meer daar tussenin ligt de koppeling via import en export van variabelen en subroutines.

b) *Complexiteit van de informatie-uitwisseling*

Deze moet gemeten worden in termen van het *aantal* items dat uitgewisseld wordt tussen modules; niet de *hoeveelheid* uitgewisselde data in bytes.

c) *Tijdstip van instantiatie van de verbinding tussen modules*

Aan compile-time geïntantieerde verbindingen wordt de voorkeur gegeven boven run-time geïntantieerde.

d) *Aard van de uitgewisselde informatie*

Op basis hiervan worden in [Gane79] drie typen koppeling onderscheiden, te weten data-coupling, control-coupling en content-coupling. De laatste wordt ook aangeduid met externe of pathologische koppeling.

De betekenis die in [Gane79] aan deze drie termen gehecht wordt komt er in essentie op neer dat data-coupling te prefereren is boven control-coupling omdat de eerste geen invloed heeft op de flow of control van de ontvangende module en de tweede wel. Men denke in dit laatste geval bijvoorbeeld aan het doorgeven van flags. Content-coupling is helemaal uit den boze. Hieronder wordt verstaan dat een module 'stiekem' de waarde van een variabele uit een andere module leest of verandert. Bij 'stiekem' kan men bijvoorbeeld denken aan een module-hiërarchie die gepasseerd wordt.

Het onderscheid tussen data en control-coupling is op deze wijze wat zwart-wit geformuleerd. Het meestal niet te vermijden dat de flow of control in de ontvangende module beïnvloed wordt. Immers, bij iedere subroutine die een test uitvoert op de waarde van een argument (bijvoorbeeld in een subroutine die wortels trekt een test of het argument al dan niet negatief is) kan men stellen dat de waarde van het argument de flow of control beïnvloedt. Deze vorm van control-coupling is echter helemaal niet slechter dan pure data-coupling mits aan twee voorwaarden is voldaan. Ten eerste moet de invloed van informatie afkomstig uit de aanroepende module op de flow of control van de aangeroepen module gespecificeerd kunnen worden zonder te refereren aan de identiteit van de aanroeper. Ten tweede mag de invloed geen effect hebben op de flow of control in de module nadat het 'resultaat' aan de aanroepende module is teruggegeven.

Zoals hierboven al is opgemerkt kan een teveel aan koppeling tussen modules in het algemeen opgevat worden als het gevolg van te weinig cohesie binnen modules.

In het algemeen onderscheidt men in software zes tot zeven niveau's van cohesie, te weten (in volgorde van afnemende wenselijkheid):

a) *Functionele cohesie*

In een functioneel coherente module is ieder element (subroutine) een integraal deel van één enkele functie. Met andere woorden, de elementen zijn zowel procedureel als ook *conceptueel* een eenheid. Als test voor functionele cohesie kan de vuistregel gebruikt worden dat de functie van de module moet zijn uit te drukken in een gebiedende wijs plus een lijdend voorwerp (bijvoorbeeld "bereken de optimale oplossing" of "formateer het antwoord").

b) *Sequentiële cohesie*

Een sequentieel coherente module is een bundeling van een aantal functioneel coherente onderdelen waarbij de uitvoer van het ene onderdeel als invoer voor een volgend onderdeel dient (bijvoorbeeld "lees invoer-regel en verwijder spaties" of "bereken oplossing en druk deze af").

c) *Communicatieve cohesie*

Een communicatief coherente module bestaat uit functioneel coherente componenten die het lijdend voorwerp gemeen hebben (bijvoorbeeld "druk de oplossing af en sla hem op in de database").

d) *Procedurele cohesie*

Procedureel coherente modules kunnen ontstaan als symbool (of meer aangrenzende de(e))l(en) van een flowchart in een module geplaatst te worden. Wanneer dit niet leidt tot *conceptuele* eenheid spreekt men van procedurele cohesie.

e) *Temporele cohesie*

De elementen (subroutines) van een temporeel coherente module hebben gemeen dat ze in hetzelfde stadium van executie plaatsvinden (bijvoorbeeld 'initialisatie').

f) *Logische cohesie*

Op elkaar gelijkende doch in detail verschillende stukken code die zijn samengevoegd met 'weglating' van identieke elementen (op één exemplaar na uiteraard) vormen een logisch coherente module (bijvoorbeeld een module die alle invoer-procedures bevat).

g) *Toevallige cohesie*

Toevallig coherente modules bestaan uit elementen waarvoor geen gemeenschappelijke noemer gevonden kan worden anders dan "opgenomen in dezelfde module".

## 5.3.2 Modulaire structurering van kennisbanken

De 'traditionele' kennisbank bestaat uit een kluwen produktieregels zonder enige modulaire structurering. Met de opkomst van tweede generatie expertsystemen begint een eerste aanzet tot structurering: meta-kennis wordt gescheiden van domeinkennis. Gaandeweg verliest de kennisbank-'programmeur' de vrijheid waar hij in het verleden het hoofd aan stootte. De noodzaak van voorstructurering van het kennismodel als specificatie van de kennisbank wordt tegenwoordig alom erkend. Er is echter nog lang geen consensus over de vorm die een dergelijke voorstructurering precies moet hebben. Ook de 'vertaling' van gestructureerde specificatie naar kennisbankstructuur is meestal niet duidelijk beschreven.

Binnen het KADS-project wordt gewerkt met een kennismodel bestaande uit drie tot vier lagen, te weten een domeinlaag, een (kanonieke) inferentie-laag, een taaklaag en soms (zelden in concrete applicaties) een strategie-laag [FEL1989-148]. Het centrale concept van KADS is dat van de generieke taak. Als voorbeeld bij uitstek mag de 'heuristische classificatie-taak' genoemd worden [Clancey85]. KADS richt zich echter met name op kennisacquisitie. Het besteedt te weinig aandacht aan andere relevante zaken om als ontwikkelmethodologie te gelden. De overgang van het kennis-model (als eindprodukt van de kennis-analyse) naar de software was aanvankelijk een stiefkind. Recentelijk is dit door formalisatie van het KADS-kennismodel aanzienlijk verbeterd [Akkermans89]. Doch beschouwingen over bijvoorbeeld testen en onderhoud komen in KADS-publicaties niet of nauwelijks aan de orde. Zulks in tegenstelling met [Chandrasekaran87], waarin

eveneens 'generieke' taken een centrale rol spelen, zij het dat de invulling van de generieke taken afwijkt. Chandrasekaran benadrukt, net als Breuker c.s., het belang van bouwstenen voor kennismodellering die van een veel hoger abstractie-niveau zijn dan zaken als produktieregels, frames en netwerken. Voortbouwend op eerder werk op het gebied van diagnostiseren en ontwerpen presenteert hij vier generieke diagnose-taken hiërarchische classificatie, hypothese-evaluatie ("in hoeverre stemmen data en hypothese overeen?"), abductief assembleren (een aantal low-level hypotheses tot een coherent geheel aan een smeden) en database inferentie [Chandrasekaran83,84,86]. Naast deze vier noemt Chandrasekaran nog twee t.b.v. ontwerp: object-synthese-middels-selectie-en-verfijning-van-plannen en toestandsruimte-abstractie. Voor alle zes taken bestaan generieke 'tools'. Chandrasekaran noemt expliciet de *onwenselijkheid* van een uniforme kennisrepresentatie voor de verschillende generieke taken. Hij ziet intelligentie als een interactie van functionele eenheden (probleem oplosers) die elk de kennisrepresentatie en inferentie-methoden gebruiken die passen bij hun aard.

Laatstgenoemde uitspraak staat dicht bij de filosofie achter blackboard systemen en object-georiënteerd programmeren. De blackboard aanpak is oorspronkelijk niet ontwikkeld om de kwaliteitsbeheersing van kennisysteemontwikkeling te verbeteren, maar om complexe problemen als spraakherkenning aan te pakken [Erman80,Fennell77]. Als zodanig is een blackboard-architectuur met name geschikt voor probleemoplossen waarbij aanvankelijk zeer onzekere hypothesen van één expert door een andere (lees: via een andere optiek) bevestigd of verworpen worden. Een dergelijke wijze van probleemoplossen is echter geschikt voor vele domeinen. Artsen maken bijvoorbeeld in hun redeneringen (bijv. diagnose) gebruik van nosologische, epidemiologische, fysiologische en anatomische kennis. Voor juristen is naast het wetboek de jurisprudentie van groot belang. Generalisaties (van domein-specifieke blackboard-systemen) als AGE/ESHELL, Hearsay-III, BB1, MXA en BLOBS vertolken de aspiratie in de richting van de algemene toepasbaarheid van blackboard-technologie [Engelmore88]. Afgezien van het gebruik van meer dan één inferentiemethode en een dosis onzekerheid in vele domeinen is er nog een ander punt dat het blackboard-paradigma tot een belangrijk concept voor kennisystemen maakt. Een blackboard-architectuur biedt goede mogelijkheden om redelijk vergaande modulariteit in te bouwen en als zodanig een overzichtelijker en beter testbaar produkt af te leveren. Dat dit aspect tot nu toe weinig aandacht gekregen heeft is waarschijnlijk te wijten aan het feit dat het ontwikkelen van



een lijvig blackboard systeem tot voor kort een hele onderneming was. Nu echter steeds meer algemene blackboard-ontwikkelomgevingen als BB1 [Hayes-Roth88] en GBB [Corkill88] beschikbaar komen, zullen blackboard-architecturen wellicht meer ingang vinden. Met name GBB (Generic BlackBoard) is vanuit het oogpunt van kwaliteitsbeheersing interessant vanwege de sterke nadruk die hierbij gelegd wordt op specificatie van de structuur voor het inbrengen van de gegevens (kennis) zelf. Hier tekent zich een duidelijke analogie of met de 'data-definitie' in database-methodologie. Bovendien wordt er in GBB veel aandacht besteed aan efficiëntie van het inbrengen van nieuwe en het opzoeken van bestaande gegevens (blackboard-objecten in dit geval). Gezien de weinig florissante reputatie die blackboard-systemen wat dit betreft hebben is dit tevens een belangrijk aspect. Een andere generalisatie die met name het nut van blackboard-technologie voor kwaliteitsbeheersing benadrukt is de Edinburgh Prolog Blackboard Shell [Engelmore88]. Als recent voorbeeld van eigen bodem van de merites van blackboard-technologie voor kwaliteitsbeheersing is het juridische kennissysteem Prolexs te noemen [Walker88].

Een andere tendens is de toenemende aandacht voor de integratie van database-technologie en kennis-technologie. Dit is bijvoorbeeld in MEDES te zien, een medisch systeem dat het midden houdt tussen een database-systeem en een ontwikkelomgeving voor medische kennissystemen [DeVriesRobb89]. De domeindeskundige werkt, aangezien hij/zij louter relatie-instantiaties kan invoeren, met software die eerder met database dan met kennissysteem aangeduid zou moeten worden. De queries die de eindgebruiker kan stellen maken echter dat het systeem voor die eindgebruiker een kennissysteem is. MEDES bestaat uit drie lagen. De binnenste is een general purpose shell. Daaromheen zit een laag met inferentie-algoritmen en predikaat-definities die voor een medisch domein (in het algemeen) van belang geacht kunnen worden. De buitenste laag bestaat uit statische domeinkennis. De domein-expert vult deze (en in principe slechts deze) laag, waarbij hij/zij alleen predikaten kan gebruiken, die in de middelste laag (de subshell geheten) gedefinieerd zijn. Aangezien alle in queries te gebruiken inferentie-primitieven gedefinieerd zijn in de subshell die voor de domein-expert niet toegankelijk is, mag van een dergelijk systeem verwacht worden dat het een snelle, soepele en niet door vervelende tikfouten bedorven implementatie van een nieuw (medisch) domein mogelijk maakt. Uiteraard levert een reeds voorgedefinieerde set van predikaten een beperking op voor de domein-deskundige die een (ander) medisch domein

moet invoeren. Ervaring zal moeten leren of dit een onoverkomelijke, hinderlijke of juist aangename beperking is. Zeker is echter dat wanneer men kennissysteemontwikkeling van 'craft' tot 'industry' wil maken een dergelijke inperking van vrijheden een goede (en wellicht de enige) weg is.

### 5.3.3 Cohesie en koppeling bij kennisbanken

Koppeling en cohesie zijn, in elk geval wat betreft de definities in [Gane86] en [Yourdon79], karakteristieken van *procedurele* software. Dat betekent echter niet dat ze geen betrekking kunnen hebben op kennisbanken: Wanneer een kennisbank in modules wordt verdeeld is het heel wel mogelijk dat een wijziging in een module wijzigingen in andere modules vereist, bijvoorbeeld om de kennisbank als geheel consistent te houden. In feite maakt het in [Nguyen87] beschreven algoritme voor consistentie-controle gebruik van een modulaire opdeling van de kennisbank om de controle efficiënter te kunnen uitvoeren, zij het dat de indeling in modules *achteraf* (d.w.z. na het ontwikkelen van het systeem) geschiedt. Omdat dit puur op basis van de afhankelijkheid tussen predikaten geschiedt zonder te letten op de conceptuele eenheid van modules is hier sprake van procedurele cohesie. Van de koppeling tussen de modules in de context-tree kan niet gezegd worden of ze data- dan wel control-gekoppeld zijn. Dat hangt ervan af of er tijdens het inferentie-proces nieuwe (afgeleide) feiten worden opgeslagen.

Bij blackboard-architecturen met een enkelvoudig (niet in panelen verdeeld) blackboard is de *koppeling* tussen de afzonderlijke kennisbanken in principe een vorm van 'common environment'-koppeling. Meestal zijn de blackboards echter wel in panelen verdeeld en is bij iedere kennisbank gespecificeerd welke panelen het mag lezen en welke het mag beschrijven. In dat geval is de koppeling eerder te omschrijven als object-georiënteerd. In beide gevallen gaat het in essentie om data-koppeling in de zin dat een verandering in een kennisbank geen ongewenste bijverschijnselen in een andere tot gevolg heeft, mits tenminste bij het ontwerp van de kennisbanken geen rekening gehouden is met de functionaliteit van die andere kennisbank. Een uitzondering hierop is het veranderen van de toegangspermissies (lezen/schrijven) van een kennisbank op de verschillende blackboard-panelen. Wanneer men ervan uitgaat dat de toegangspermissies een overzichtelijk geheel vormen zal dit geen problemen geven. Overigens kan een dergelijke verandering gezien worden als een verandering van de globale specificatie, hetgeen bij

normaal onderhoud niet zal optreden. Een blackboard-systeem als Hearsay-II zijn de afzonderlijke kennisbanken niet alleen procedureel, maar ook conceptueel duidelijk te onderscheiden. Er is dus sprake van *functionele cohesie*. Van een *module-hiërarchie* is bij 'gewone' blackboard-systemen geen sprake. Gedistribueerde blackboard-systemen bezitten een beperkte (twee-lagige) module hiërarchie [Durfee88].

#### 5.4 Database-technologie en kwaliteitsbeheersing

##### 5.4.1 Integriteit in databases en knowledgebases

Uit paragraaf 5.2 moge duidelijk geworden zijn dat verantwoord testen niet alleen bij kennissystemen maar ook bij databases met louter conventionele middelen (d.w.z. equivalentie-klassen en coverage) geen haalbare kaart is. Bovendien is de data in een database aan veelvuldige veranderingen onderhevig. Het handhaven van integriteitsbeperkingen is voor databases een nuttige en nodige aanvulling. Er zijn vele soorten integriteitsbeperkingen bij databases (zie hoofdstuk 4), maar ze hebben met elkaar gemeen dat ze relatie- of object-gebonden zijn. (Dit rapport beperkt zich tot relationele databases.)

Zoals reeds in het fase 1-rapport is opgemerkt zijn integriteitsbeperkingen in kennissystemen, als ze al voorkomen, van algemenere (d.w.z. minder predikaat-gebonden aard) dan integriteitsbeperkingen in databases [FEL1989-148]. In de meeste publikaties spreekt men slechts over de consistentie van kennisbanken [Bezem87]. Af en toe komt ook de volledigheid van kennisbanken aan de orde [Suwa85,Marek87]. En een enkeling bespreekt daarnaast nog andere eigenschappen die als concretisatie van de 'correctheid' van een kennisbank gebruikt kunnen worden [Nguyen85,87]. Al deze eigenschappen zijn echter karakteristieken van de kennisbank als geheel. In tegenstelling tot database-integriteit is 'kennisbankintegriteit' geen flexibel kader waarin de programmeur (kennisingenieur) zelf kan aangeven welke integriteitsbeperkingen gelden. Kennisbank-integriteit, zoals het tot op heden is vormgegeven via de begrippen consistentie en volledigheid, redundantie, subsumptie en circulariteit is ook in eerste instantie bedoeld als debugging-tool. Ondersteuning van het onderhoud zoals bij databases in de vorm van integriteitscontrole bij transacties ('updates') plaatsvindt komt hierbij pas op de tweede

plaats. De huidige vormgeving van kennisbankintegriteit schiet te kort op een aantal punten, te weten:

1. Er is geen methode, die zowel theoretisch goed als praktisch efficiënt is; Het conceptuele kader is in [Bezem87] helder neergezet, maar de methode is hier onvoldoende efficiënt om van praktisch nut te zijn in kennisbanken waarin produktieregels variabelen kunnen bevatten. Bovendien beperkt Bezem zich tot consistentie (in de strikte, logische zin). De andere genoemde artikelen zijn conceptueel niet erg helder. Ook [Marek87], dat vanuit een database-perspectief algoritmes voor consistentie en volledigheid te geven schiet in helderheid te kort. Bovendien is diens algoritme voor volledigheidscntrole gebouwd op de onjuiste vooronderstelling dat een kennisbank volledig is (alle attribuutwaarden bestrijkt) dan en slechts dan als de afzonderlijke relatie-typen een volledige extensie hebben. Een tegenvoorbeeld is simpel:

---

KB1: Als  $x < 0$  EN  $y \geq 0$  DAN a1  
Als  $x \geq 0$  EN  $y \geq 0$  DAN a2

KB2: Als  $y < 0$  EN  $z \geq 0$  DAN a3  
Als  $y < 0$  EN  $z < 0$  DAN a4

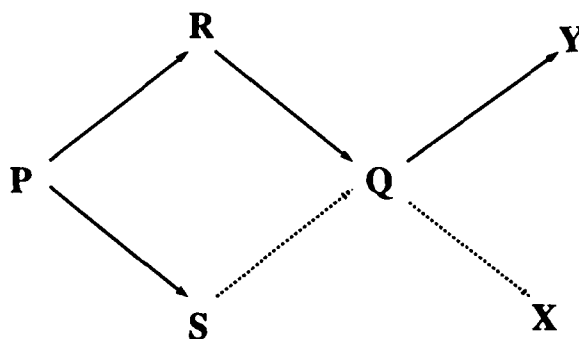
---

Figuur 5.2: Voorbeeld van een kennisbank.

KB1 is onvolledig ten opzichte van de attributen x en y en KB2 idem dito ten opzichte van y en z. Maar KB1 + KB2 is wel volledig ten opzichte van x, y en z.

2. Het merendeel van de in [Nguyen85] genoemde integriteitsbeperkingen valt in het water als onzekerheidsmaten (Bayesiaanse maten, zekerheidsfactoren e.d.) om de hoek komen kijken. (Dit is uitgewerkt in [Nguyen87b].) Men zou zich kunnen laten verleiden om te streven naar aangepaste algoritmes voor integriteitscontrole, maar het is waarschijnlijk dat een strakkere regelgeving voor het ontwerp van kennisbanken met onzekere inferentie nodig is om tot enigszins betrouwbare en onderhoudbare software te komen. Onzekere inferentie nodigt in zekere zin uit om maar wat aan te rommelen, 'want het komt toch allemaal niet zo precies'. Het volgende voorbeeld kan wellicht illustreren hoe gemakkelijk onzorgvuldigheden over het hoofd worden gezien.

Beschouw bijvoorbeeld de beïnvloedingsgraaf in figuur 5.3: Van de variabelen P naar Q lopen twee pijlen (=beïnvloedingen) waar- van de een via R (potentieel) stimulerend en de ander via S (potentieel) remmend. (noot: De 'stimulerende invloed' van R op Q betekent in concreto dat verhoging van R verhoging van Q tot gevolg kan hebben en verlaging van R verlaging van Q. 'Remmende invloed' is analoog gedefinieerd) Q heeft een remmende invloed op X en een stimulerende op Y. Een vraag aan het systeem zou kunnen luiden: "X en Y zijn beide verhoogd. Zou een verhoogde P hiervan de oorzaak kunnen zijn?"



Figuur 5.3: Voorbeeld van een beïnvloedingsgraaf.

Het 'beïnvloedings-teken' langs een pad ( stimulerend = + ; remmend = - ) kan middels tekenvermenigvuldiging van de schakels ervan worden bepaald. Er is dan uitgaande van P naar X zowel een stimulerend (via S) als een remmend (via R) pad. Idem dito voor de invloed van P op Y. Een te naïef algoritme gaat nu simpelweg van X en Y afzonderlijk na of er een stimulerend pad bestaat van P naar X respectievelijk Y en komt dus met een bevestigend antwoord. "Ja, P kan de oorzaak zijn van een (gelijktijdige) verhoging van zowel X als Y." Dit is onjuist, want alle invloeden gaan via Q , die een tegengestelde invloed heeft op X en Y. In dit geval moet dus in het predikaat 'verklaart' worden geïncorporeerd dat de variabelen die de gebruikte paden gemeen hebben via de verschillende paden eenzelfde waarde moeten krijgen toebedeeld.

3. De begrippen consistentie en volledigheid bestrijken slechts een zeer klein gedeelte van de mogelijke integriteitsbeperkingen. Ook wanneer redundantie, subsumptie en circulariteit eraan toegevoegd worden blijft het nog een mager geheel in vergelijking met het 'normale' arsenaal aan integriteitsbeperkingen bij databases. Dit is des te schrijnender omdat veel kennissystemen eigenlijk weinig verschillen van databases, in de zin dat ze veel meer feiten dan produktieregels bevatten en recursieve regels zelden voorkomen.

#### 5.4.2 Integriteit in knowledgebase-management-systemen

Databases zijn 'ouder' dan kennissystemen en dientengevolge is het beheersen van de integriteit in databases meer ontwikkeld dan de beheersing van consistentie en volledigheid bij kennissystemen. De zogenaamde expert-database-systemen ofwel knowledge-base-management-systemen lijken qua implementatie meer op een database, doch qua buitenaanzicht (en deductieve vermogens) meer op een kennisstelsel. Voor deze systemen verricht men al enige tijd onderzoek naar een krachtige(r) integriteitsbewaking, zowel wat de uitdrukingskracht van de beperkingsregels als de efficiëntie van het effectueren ervan betreft.

In [Decker87] wordt een reeds bekende simplificatiemethode van [Nicolas82] voor beperkingsregels op 'gewone' databases gegeneraliseerd naar deductieve databases (d.w.z. databases waarbij deductieve queries gesteld kunnen worden). Wat

uitdrukkingskracht betreft: het algoritme van Nicolas en ook dat van Decker kan een subklasse van de predikaatlogica aan die op te vatten is als meersoortige logica, de zogenaamde 'range-restricted formulas'. Range-restricted komt op hetzelfde neer als 'safe': Het drukt uit dat voor de evaluatie van een formule alleen van belang is wat er in de database staat [Ullman80]. De onwaarheid van een existentiële uitspraak volgt zo uit de afwezigheid van het betreffende feit in de database. Evenzo kan tot de geldigheid van een universele uitspraak worden besloten op grond van de inhoud van de database. In de AI-wereld staat een dergelijke aanname bekend als de 'Closed World Assumption'. Wat de efficiëntie betreft, komt de aanpak van Nicolas erop neer dat het feit dat de database voor een transactie aan de integriteitsbeperkingen gehoorzaamt te samen met het type van de transactie (insert, delete, update) benut wordt om de oorspronkelijke, declaratieve integriteitsbeperking te vereenvoudigen tot een sneller evalueerbare expressie.

Decker heeft het algoritme van [Nicolas82] gegeneraliseerd van databases naar deductieve databases, dus 'databases' waarin behalve feiten ook regels gemanipuleerd kunnen worden. Bovendien worden de beperkingsregels compile-time vereenvoudigd, wat een fikse besparing oplevert. Sadri en Kowalski gaan in [Sadri88] nog een stapje verder en presenteren een algoritme dat afhankelijk van een parameter-instelling dat van [Decker86] of dat van [Lloyd86] kan imiteren. Het is in feite een theorem-proving-aanpak; een variant op de bekende (Prolog) SLDNF-resolutie. Er wordt hierbij zowel achterwaarts als voorwaarts geredeneerd.

#### 5.4.3 POSTGRES als knowledgebase-management-systeem

Kennissystemen in het algemeen steken wat betreft kwaliteitsbeheersing en daartoe ontwikkelde methoden en technieken schril af tegen databases: Integriteitsbeperkingen en efficiënte algoritmes voor de handhaving van integriteit zijn slechts rudimentair aanwezig. Een analogon van de normaalvormen als richtlijnen voor database-ontwerp ontbreekt bij kennissystemen zelfs volledig. En general purpose shells met een verspreiding die te vergelijken is met standaard database systemen als ORACLE en INGRES zijn er niet. De recente ontwikkelingen op het gebied van de incorporatie van AI-technieken in databases scheppen interessante perspectieven. Een van de belangrijkste aspecten die men als 'het' verschil tussen kennissystemen en databases zou kunnen aandragen is het ontbreken van recursie in de queries van databases. Hier wordt echter

hard aan gewerkt. In [Vieille88] wordt een theoretisch raamwerk voor het verwerken van recursieve queries en views door een deductief database systeem gepresenteerd. In tegenstelling tot Prolog kan het van SLD-resolutie afgeleide algoritme (positieve) cyclische afhankelijkheden aan.

Naast recursieve queries wordt er ook enige aandacht besteed aan modulariteit. Een beproefd database-systeem als ORACLE kent geen segmentatie. In het hieronder besproken POSTGRES-systeem is het *wel* mogelijk de database te segmenteren.

Methoden en technieken die in het kader van knowledgebase-management-systemen (KBMS) zijn ontwikkeld, hebben de potentie om kwaliteitsproblemen voor een grote klasse van kennissystemen geheel of gedeeltelijk uit de wereld te helpen. Er zijn diverse voorstellen gedaan met betrekking tot de architectuur van KBMS'en [Brodie89, Stonebraker89]. Meestal brengt men een onderscheid aan tussen 'loosely-coupled' en 'tightly-coupled' systemen. Onder een 'loosely-coupled' systeem wordt verstaan een extern database-management-systeem (DBMS) dat via een interface gekoppeld is aan een logische programmeertaal: bijvoorbeeld het database-systeem Oracle gekoppeld aan de taal Quintus Prolog. In het geval van een 'loosely-coupled' KBMS bestaat er een programma, bijvoorbeeld geschreven in Prolog, dat kennisregels bevat op een bepaald gebied. De regels zijn op te vatten als views op een interne danwel externe database. Prolog-regels kunnen in deze configuratie tot queries op de database aanleiding geven. Een gebruiker van het Prolog programma hoeft zich niet bewust te zijn van de verschillen tussen intern en extern geheugen. Door middel van een interface zullen de gewenste gegevenselementen worden opgehaald [Quintus89].

In een 'tightly-coupled' systeem wordt geen onderscheid meer gemaakt tussen een database-systeem en een logische taal. Dit kan enerzijds worden gerealiseerd door een logische programmeertaal uit te breiden met database-faciliteiten zoals integrity, concurrency, security en recovery. Anderzijds is het mogelijk om een database uit te breiden met deductieve faciliteiten zoals in een logische programmeertaal [Stonebraker88]. Een interessant voorbeeld van het laatstgenoemde alternatief is POSTGRES, een verdere ontwikkeling van het relationele database-systeem INGRES (POST inGRES) [Stonebraker86].



POSTGRES is een 'tightly-coupled' knowledgebase-management-systeem dat ontwikkeld wordt aan de Universiteit van Zuid-Californië te Berkeley. De belangrijkste doelstellingen zijn het zoveel mogelijk handhaven van het relationele model en het bieden van faciliteiten voor 'actieve' databases en inferentie, waaronder voorwaarts en achterwaarts redeneren. In veel toepassingen is het handig om te kunnen werken met triggers en alerters. Triggers zijn kleine stukjes SQL-programma (Structured Query Language: de gestandaardiseerde vraagtaal voor relationele DBMS'en) die kunnen worden geactiveerd nadat veranderingen in de database zijn aangebracht (insert, delete en update). Alerters zijn te vergelijken met triggers, maar worden geactiveerd door tijd of datum. Echter, het gebruik van regels (en procedures) alsof het gegevenselementen zijn is wel de meest revolutionaire ontwikkeling in POSTGRES. Alle voordelen die een database-systeem heeft als het gaat om integrity, concurrency, security en recovery worden nu uitgebreid naar het werken met regels. Volgens de definitie van Nijssen is een kennissysteem op te vatten als een systeem dat bepaalde menselijke expertise bevat en bestaat uit een verzameling aan elkaar gerelateerde feiten [Nijssen86]. De feiten kunnen worden ingevoerd door de gebruiker, of worden door het systeem zelf afgeleid op basis van andere feiten en inferentieregels. Regels in POSTGRES kunnen leiden tot voor- en achterwaarts redeneren. Dit wordt bereikt met zogenaamde 'vroeg' en 'late' evaluatie van regels. In het geval van 'vroeg' evaluatie zal een verandering in een gegevenselement dat opgenomen is in een regel direct tot gevolg hebben dat de regel wordt uitgevoerd (geëvalueerd). Bij 'late' evaluatie wordt een verandering in een gegevenselement pas waargenomen op het moment dat het wordt opgevraagd [Stonebraker88].

Het is niet de bedoeling om een uitgebreide beschrijving van het nog in ontwikkeling zijnde POSTGRES-systeem te geven. In deze paragraaf worden slechts enkele saillante punten naar voren gehaald die van direct belang zijn bij de ontwikkeling van een kennissysteem, met name de behandeling van inferentieregels.

Door in een DBMS/KBMS gebruik te maken van inferentieregels is het mogelijk om runtime een relatie te 'berekenen'. M.a.w. het systeem beschikt over de intensie van de toepassing (de definitie van tabellen, de onderlinge afhankelijkheden en te gebruiken inferentieregels) en bepaalt op het moment dat daar behoefte aan is een extensie (de werkelijke feiten in de tabellen). Stel men beschikt over een relatie waarin is aangegeven

dat persoon A een ouder is van persoon B. Als men nu wil vaststellen wie de voorouders van een willekeurige persoon C zijn, dan moet een relatie worden opgebouwd die de betreffende feiten weergeeft. In Prolog is dit als volgt te beschrijven:

---

```
voorouder(Voorouder,Afstammeling):-
 ouder(Voorouder,Afstammeling).
voorouder(Voorouder,Afstammeling):-
 ouder(Voorouder,Persoon),
 voorouder(Persoon,Afstammeling).
```

---

Figuur 5.4: Voorbeeld van een Prolog-programma.

Dit klassieke probleem is een voorbeeld van achterwaarts redeneren. Immers, om te weten te komen wie allemaal voorouders van een bepaalde persoon zijn, wordt eerst vastgesteld wie de ouder is, vervolgens wie de ouder van de ouder is enzovoort totdat de verzameling van voorouders niet verder meer wordt uitgebreid. Door gebruik te maken van het view-mechanisme in een DBMS is dit redeneergedrag te simuleren. Het komt neer op een query die, evenals in het Prolog voorbeeld, is gebaseerd op de OUDER-relatie. Uit het Prolog programma blijkt dat het een recursieve query betreft. In SQL, of het Ingres derivaat QUEL, is het niet mogelijk een recursieve query op te stellen. POSTQUEL, de vraagtaal van POSTGRES, biedt deze mogelijkheid wel; door achter bepaalde keywords als APPEND, DELETE, EXECUTE, DEFINE VIEW, REPLACE EN RETRIEVE een asterisk (\*) te plaatsen wordt aangegeven dat de query herhaald moet worden uitgevoerd. De query 'stopt' op het moment dat een relatie niet verder meer verandert. Nu wordt uitgegaan van een bestaande OUDER-relatie: OUDER(ouder\_van,afstammeling). De VOOROUDER-relatie kan nu als volgt worden gedefinieerd:

---

```
range of O is OUDER
range of VO is VOOROUDEr
define view VOOROUDEr(O.all)
define view* VOOROUDEr(VO.ouder_van, O.afstammeling)
 where VO.afstammeling = O.ouder_van
```

---

Figuur 5.5: Voorbeeld van een recursieve POSTQUEL-query.

In dit POSTQUEL-programma wordt in de VOOROUDEr-relatie (die gedefinieerd is als een 'view' op de gegevensbank) eerst een kopie geplaatst van alle feiten in de OUDER-relatie: dit zijn de directe afstammelingen. Daarna wordt recursief bepaald welke andere, indirecte relaties er bestaan. Nu kan een query opgesteld, die tot gevolg heeft dat de VOOROUDEr-relatie met behulp van recursieve aanroepen wordt opgebouwd:

---

```
retrieve (VOOROUDEr.ouder_van)
 where VOOROUDEr.afstammeling = "Klaas"
```

---

Figuur 5.6: Opbouw van de voorouder-relatie.

Naast achterwaarts redeneren is het ook mogelijk om voorwaarts te redeneren. In dit geval wordt gebruik gemaakt van triggers, stukjes POSTQUEL-code die worden geactiveerd als zich een verandering voordoet in de database/knowledgebase. Stel dat Piet's salaris altijd gelijk is aan dat van Klaas, dan moet een verandering in Klaas' salaris altijd leiden tot een verandering in het salaris van Piet. M.a.w. in POSTQUEL:

---

```
always replace MEDEWERKER(salaris = MED.salaris)
 using MED in MEDEWERKER
 where MED.naam = "Klaas" and MEDEWERKER.naam = "Piet"
```

---

Figuur 5.7: Voorwaarts redeneren in POSTQUEL.

Het keyword 'always' geeft aan dat deze verandering direct wordt doorgevoerd.

De gegeven (kleine) voorbeelden kunnen naar believen worden uitgebreid. Meer technische vragen als: wat gebeurt er als in een bepaalde situatie meer dan een regel van toepassing is, zijn opgelost door aan regels prioriteiten mee te geven. Een voorbeeld van een POSTQUEL-programma dat voor elke twee punten in een graaf het pad met de minimale kosten berekend is hieronder weergegeven. De relatie BEREIKBAAR(begin,eind,kosten) is al in de database aanwezig:

---

```
range of B1 is BEREIKBAAR
range of B2 is BEREIKBAAR
retrieve into TEMP(B1.begin,B2.eind,kosten=ONEINDIG)
 where B1.begin /= B2.eind
replace TEMP(kosten=BEREIKBAAR.kosten)
 where TEMP.begin=BEREIKBAAR.begin
 and
 TEMP.eind=BEREIKBAAR.eind
```

---

Figuur 5.8.1: Zoeken in een graaf met POSTQUEL.

---

```
range of T is TEMP
replace* TEMP(kosten=T.kosten+BEREIKBAAR.kosten)
 where T.kosten+BEREIKBAAR.kosten < TEMP.kosten
 and
 T.eind=BEREIKBAAR.begin
 and
 BEREIKBAAR.eind=TEMP.eind
```

---

Figuur 5.8.2: Zoeken in een graaf met POSTQUEL.

In de kunstmatige intelligentie wordt veel aandacht besteed aan zoekmethoden. Algoritmes als A\* en AO\* zijn bekende voorbeelden, die gebruik maken van heuristische schattingen van de 'lengte' van nog af te zoeken paden in een graaf. Tijdens een zoekproces kunnen verschillende keuzes worden gemaakt voor wat betreft de verder te beschouwen knooppunten in de graaf; welke moeten als eerste worden uitgebreid? Hiervoor wordt vaak gekozen uit 'best-first' en 'breadth-first' algoritmes. In het eerste geval worden slechts die knooppunten uitgebreid die op het kortste pad naar het uiteindelijke doel liggen, wat wordt gerepresenteerd door de geringste kosten (afgelegde afstand) tot dan toe; de besten eerst. In het tweede geval worden na elke stap in het zoekproces alle op hetzelfde niveau gegenereerde knooppunten uitgebreid; eerst in de breedte zoeken. Een voordeel van 'breadth-first' zoeken is dat altijd het kortste pad tussen de uitgangssituatie en de doelsituatie (de oplossing van het probleem) wordt gevonden. Een nadeel is echter dat het in de breedte zoeken een veel geheugen vergende activiteit is. Met dit laatste probleem wordt men bij het 'best-first' algoritme niet geconfronteerd, maar daar staat tegenover dat de keuze van het beste pad op een bepaald niveau niet noodzakelijkerwijze hoeft te betekenen dat alle hierna volgende knooppunten ook op dit beste pad liggen.

In een KBMS als POSTGRES kunnen zoekmethoden worden gerepresenteerd. Om een en ander te verduidelijken een voorbeeld. Het probleem is weer het vinden van een

kortste route tussen twee punten in een graaf. Er wordt weer gebruik gemaakt van de relatie `BEREIKBAAR(begin,eind,kosten)`. Tevens is de relatie `TOESTANDEN(eind,kosten)` in gebruik, om de toestand van het systeem bij te houden tijdens de werking van het proces. Hier wordt onder kosten verstaan de kosten van het kortste pad tot dusver.

Allereerst een weergave van de 'breadth-first' zoekmethode:

---

range of S,T is `TOESTANDEN`

range of B is `BEREIKBAAR`

retrieve into `KNOOPPUNTEN(S.eind)`

range of K in `KNOOPPUNTEN`

append to `KNOOPPUNTEN(K.eind,kosten=ONEINDIG)`

replace `S(kosten=0)` where `S.eind=BEGIN`

replace\* `T(kosten=S.kosten+B.kosten)` where

`B.begin=S.eind` and `B.eind=T.eind`

and

`T.kosten>B.kosten+S.kosten`

---

Figuur 5.10: 'Breadth-first' zoekmethode in POSTQUEL.

Vervolgens de 'best-first' zoekmethode. Aan de `TOESTANDEN`-relatie is een attribuut toegevoegd dat aangeeft of het betreffende knooppunt al is geëxpandeerd (ofwel 'open' is):

---

range of S,T is TOESTANDEN  
range of B is BEREIKBAAR

retrieve into KNOOPPUNTEN(S.eind)  
range of K is KNOOPPUNTEN  
append to TOESTANDEN(K.eind,kosten=ONEINDIG,open=true)  
replaces(kosten=0) where S.eind=BEGIN  
append to BEREIKBAAR(K.eind,K.eind,0)

execute\*

```
{ retrieve(minkosten=min(S.kosten where S.open=true))
 retrieve(mineind=min(S.eind where S.open=true and S.kosten=minkosten))

 replace T(open=(T.kosten/=minkosten),kosten=S.kosten+B.kosten)
 where
 S.eind=mineind
 and
 B.begin=mineind and B.eind=T.eind
 and
 (T.kosten>B.kosten+S.kosten
 or
 T.eind=S.eind
)
}
```

---

Figuur 5.11: 'Best-first' zoekmethode in POSTQUEL.

Naast de genoemde vernieuwingen biedt POSTGRES ook ondersteuning bij het representeren van complexe objecten. Hierbij kan gewerkt worden met hiërarchieën die veel weg hebben van semantische netwerken. Acties als 'generaliseren' of 'specificeren'

zijn dan eenvoudig uit te voeren. Een ander aspect is de mogelijkheid om zelf een bepaalde access-methode aan het systeem toe te voegen. Naast de veelgebruikte 'B-boom' methode (in ORACLE en INGRES) kunnen bijvoorbeeld ook de 'R-boom' en 'K-D-B-boom' worden toegepast [Stonebraker86]. Hierdoor kunnen feiten met polygoon-gegevens worden verwerkt, wat zeer interessant is bij het opslaan van landkaarten. Hoewel de hierboven besproken ontwikkelingen hun relevantie in de praktijk nog moeten aantonen bieden ze op zijn minst een interessant perspectief op het tot stand komen van een kennistechnologie waarvoor de kwaliteitsbeheersing uitstijgt boven de huidige mogelijkheden.



## 6 CONCLUSIES EN AANBEVELINGEN

Ervan uitgaande dat van alle kwaliteitscriteria die men aan kennisysteemontwikkeling kan verbinden juist diegene het belangwekkendst zijn die voor kennisystemen problemen blijken op te leveren en voor kennisystemen een andere uitwerking behoeven dan voor andere software komen betrouwbaarheid en onderhoudbaarheid als belangrijkste criteria naar voren.

De oplossing voor bovengenoemde, in de praktijk gesignaleerde problemen moet met name gezocht worden in het verbeteren van de modulariteit en integriteit van kennisbanken.

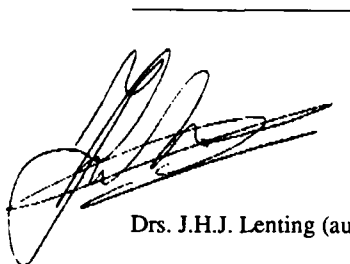
Twee technologieën binnen de software-engineering lijken goede papieren te hebben om deze verbeteringen te helpen realiseren. De database-technologie is met name in staat om de benodigde concepten en methoden voor integriteitsbewaking te leveren. Weliswaar kunnen de voor kennisbanken benodigde integriteitsbeperkingen niet allemaal teruggevonden worden bij conventionele database-systemen, maar de voor integriteitsbewaking in *deductieve* database-systemen geconcipeerde methoden en technieken strekken dusdanig ver dat ook de onder de noemer "consistentie en volledigheid" te rangschikken integriteitsbeperkingen eronder vallen. Daarnaast biedt het gebruik van database-technologie aanvullende voordelen wat betreft security, recovery, concurrency en gedistribueerde systemen.

Voor het aanbrengen van een modulaire structuur biedt de database-technologie wat minder mogelijkheden, hoewel de bij het nieuwe POSTGRES-systeem gecreëerde mogelijkheid tot segmentering zeker een stap in de goede richting is. Wat modulaire structuur betreft lijkt de blackboard-technologie echter het betere alternatief te zijn. Overigens is het opvallend dat in artikelen uit deze hoek veelal gesproken wordt van de *databases* van het systeem (in plaats van knowledge-bases).

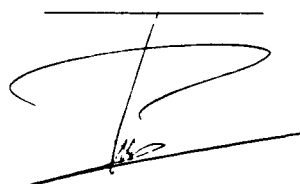
Wat betreft het toepassingsgebied kan gesteld worden dat de database-aanpak met name vruchten zal afwerpen bij domeinen waarin het leeuwendeel van de kennis uit feiten bestaat die in een niet al te groot aantal verschillende bestanden worden ondergebracht.

Bij een relatief klein aantal bestanden (*relatie-typen*) zal het aantal afleidings- en beperkingsregels ook niet al te groot zijn zodat modulaire structurering van die regels (waar binnen database-technologie niet of nauwelijks ondersteuning voor geboden wordt) niet nodig is om het systeem overzichtelijk te maken. In principe is de gespiegelde situatie (betrekkelijk weinig feiten, doch veel inferentieregels) een indicatie om soelaas te zoeken bij blackboard-technologie. Hoewel deze in eerste instantie bedoeld is voor het ontwikkelen van kennissystemen in domeinen waar meerdere, goed te onderscheiden kennisbronnen samen tot een oplossing moeten zien te komen, kan zij ook in domeinen waarin dat niet of nauwelijks het geval is tot een helderder ontwerp en daarmee tot kwaliteits-verbetering leiden.

Hoewel deze technologieën in de meeste gevallen een flinke kwaliteitsverbetering zullen kunnen bewerkstelligen zijn ze niet bij machte *alle* kwaliteitsproblemen van kennissysteemsoftware uit de wereld te helpen. Met name de kennismodellering zal voor bepaalde, als "pathologisch" aan te merken domeinen altijd een probleem blijven. Hier wordt gedoeld op de zogenaamde pré-paradigmatische kennisdomeinen, dat wil zeggen die domeinen die (nog) niet op enigszins overzichtelijke wijze van een expliciete theorie zijn voorzien. In concreto kan men hierbij denken aan sociaal-wetenschappelijke en medische domeinen, waar concepten vaak niet scherp gedefinieerd zijn en een veelheid van synoniemen en bijna-synoniemen het zicht verder vertroebelt. Bij dergelijke domeinen staat de kennisingenieur voor de taak zelf een coherente en compacte theorie op te stellen. Het hoeft geen betoog dat voor een dergelijke *essentieel creatieve* taak hooguit enige ondersteuning doch geen pasklare oplossing - van welke technologie dan ook - mag worden verwacht. Ook aan ondersteuning hebben noch database- noch blackboard-technologie wat dit betreft echter veel te bieden. In sommige gevallen kan een vertaling van concepten uit een ander domein (het paradigma van KADS) wellicht uitkomst bieden. Voor de gevallen waarin dit geen goede oplossing biedt zijn de binnen het kennisgrafenproject aan de Universiteit van Twente ontwikkelde concepten en methoden interessant. Deze zijn echter nog te weinig aan de praktijk getoetst om met zekerheid iets over de toepasbaarheid ervan te kunnen zeggen.

A handwritten signature in black ink, consisting of several loops and a long horizontal stroke at the end.

Drs. J.H.J. Lenting (auteur)

A handwritten signature in black ink, featuring a large 'P' and a horizontal line at the bottom.

Drs. M. Perre (auteur)

A handwritten signature in black ink, with a horizontal line at the top and a long horizontal stroke at the bottom.

Ir. J. Bruin (projectleider)

## 7 REFERENCES

- [Akkermans89] Akkermans, J.M. (e.a.), "Naar een Formele Specificatie van Kennismodellen", In: "Proceedings NAIC '89", 1989, pp. 105-119.
- [Atzeni88] Atzeni, P.en D.S. Parker Jr., "Formal Properties of Net-based Knowledge Representation Schemes", In: "Data & Knowledge Engineering", Vol. 3, 1988.
- [Bakker87] Bakker, R.R., "Knowledge Graphs: Representation and Structuring of Scientific Knowledge", Universiteit Twente, 1987.
- [Bezem87] Bezem, M., "Consistency of Rule-Based Expert Systems", Report CS- R8736, CWI, Computer Science/Department of Software Technology, 1987.
- [Boehm88] Boehm, B.W., "A Spiral Model of Software Development and Enhancement", In: "IEEE Computer", Mei 1988, .
- [Breuker87] Breuker, J. (ed.), "Model-Driven Knowledge Acquisition: Interpretation Models", University of Amsterdam, 1987.
- [Brodie89] Brodie, M.L. (e.a.), "Future Artificial Intelligence Requirements for Intelligent Database Systems", In: Kerschberg, L. (ed.), "Expert Database Systems, Proceedings from the Second International Conference", 1989, pp. 45-62.
- [Bruin88] Bruin, J.(e.a.), "Damocles, een Relationeel Expertsysteem", In: "Proceedings AI Toepassingen '88, 1988, pp. 217-219.
- [Chandrasekaran83] Chandrasekaran, B., "Towards a Taxonomy of Problem Solving Types", In: "AI Magazine", Vol. 4, Nr. 1, 1983.
- [Chandrasekaran84] Chandrasekaran, B., "Expert Systems: Matching Techniques to Tasks", In: Reitman W. (Ed.), "Artificial Intelligence Applications for Business", Ablex, 1984.
- [Chandrasekaran86] Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design", In: "IEEE Expert", Vol. 1, Nr. 3, 1986, pp. 23-30.
- [Chandrasekaran87] Chandrasekaran B., "Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks", In: "Proceedings IJCAI", 1987, pp. 1183-1192.
- [Clancey85] Clancey, W.J., "Heuristic Classification", In: "Artificial Intelligence", Vol. 27, 1985, pp. 289-350.
- [Conklin87] Conklin, J., "Hypertext: A survey and Introduction", In: "IEEE Computer", Vol. 20, Nr 9, 1987.

- [Corkill86] Corkill, D.D., Gallagher, K.Q., Murray, K.E., "GBB: A Generic Blackboard Development System", In: "Proceedings AAAI", 1986, pp. 1008-1014.
- [Cullen88] Cullen, J. en A. Bryman, "The Knowledge Acquisition Bottleneck: Time for Reassessment?", In: "Expert Systems", Vol. 5, Nr. 3, Augustus 1988.
- [DeVriesRobbé89] De Vries Robbé P.F., Zandstra P.E., Beckers W.P.A., "Verschillende redeneermechanismen in MEDES", In: "Proceedings NAIC-89", Academic Service, 1989, pp. 59-68.
- [Decker88] Decker, H., "Integrity Enforcement on Deductive Databases", In: Kerschberg, L (ed.), "Expert Database Systems, Proceedings of the First International Conference on Expert Database Systems", 1988, pp. 381-395.
- [Deutsch82] Deutsch, M.S., "Software verification and validation: Realistic project approaches", 1982.
- [Doyle88] Doyle, J., "Expert Systems and the 'Myth' of Symbolic Reasoning", In: "IEEE Transactions on Software Engineering", Vol. SE-11, Nr. 11, November 1988.
- [Durfee88] Durfee, E.H., "Coordination of Distributed Problem Solvers", Kluwer Academic Publishers, 1988.
- [Engelmore88] Engelmore, R.S., Morgan A.J., "Blackboard Systems", Addison-Wesley, 1988.
- [Erman80] Erman, L.D., Hayes-Roth F.A., Lesser V.R., Raj Reddy D., "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", In: "Computing Surveys", Vol. 12, Nr. 2, 1980, pp. 213-253.
- [FEL1989-148] Dekker, S.T., Lenting J.H.J., Perre M., Rutten J.J.C.R., "Kwaliteit van Expertsystemen: Een Oriëntatie", FEL 1989-148, 1989.
- [FEL-89-A267] Lenting, J.H.J. en M. Perre, "Kwaliteit van Expertsystemen: Methoden en Technieken", FEL-89-A267, 1989.
- [Fennell77] Fennell, R.D., Lesser V.R., "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay-II", In: "IEEE Transactions on Computers", 1977, pp. 98-111.
- [Frost86] Frost, D., "Introduction to Knowledge Based Systems", Collins, 1986.
- [Gane79] Gane, C. en T. Sarson, "Structured Systems Analysis", Prentice-Hall, 1979.
- [Gibson89] Gibson, V.R., Senn, J.A., "System Structure and Software Maintenance Performance", In: "Communications of the ACM", Vol. 32, Nr. 3, 1989.
- [Hayes-Roth88] Hayes-Roth B., Hewett M., BB1: "An Implementation of the Blackboard Control Architecture", In: Engelmore R.S., Morgan A.J. (Eds), "Blackboard Systems", Addison-Wesley, 1988.

- [Kuhn62] Kuhn, T.S., "The Structure of Scientific Revolutions", University of Chicago, 1962.
- [Llinas87] Llinas, J., Rizzi, S., "The Test and Evaluation Process for Knowledge Based Systems", Technical Report F30602-85-C-0313, Calspan Corporation, June, 1987.
- [Marek87] Marek, W., "Completeness and Consistency in Knowledge Base Systems", In: Kerschberg L. (Ed), "Expert Database Systems", Benjamin/Cummings, 1987.
- [Myers79] Myers, G.J., "The Art of Software Testing", Wiley, 1979.
- [Napheys89] Napheys, B. en D. Herkimer, "A Look at Loosely-Coupled Prolog Database Systems", In: Kerschberg, L. (ed.), "Expert Database Systems, Proceedings from the Second International Conference", 1989, pp. 257-271.
- [Newell82] Newell, A., "Knowledge Level", In: "Artificial Intelligence", Vol. 18, 1982.
- [Nguyen85] Nguyen, T.A., "Checking an Expert System Knowledge Base for Consistency and Completeness", In: "Proceedings of the Ninth International Joint Conference on AI", Vol. 1, 1985, pp. 375-378.
- [Nguyen85] Nguyen, T.A., Perkins, W.A., Laffey, T.J., Pecora, D., "Checking an Expert Systems Knowledge Base for Inconsistency and Completeness", In: "Proceedings 9th IJCAI", Vol. 1, 1985, pp. 375-378.
- [Nguyen87] Nguyen, T.A., "Verifying Consistency of Production Systems", In: "Proceedings of the IEEE 3rd Conference on AI", 1987, pp. 4-8.
- [Nguyen87b] Nguyen, T.A., Perkins W.A., Laffey T.J., Pecora D., "Knowledge Base Verification", In: "AI Magazine", 1987, pp. 69-75.
- [Nicolas82] Nicolas, J.M., "Logic for Improving Integrity Checking in Relational Data Bases", In: "Acta Informatica", Vol. 18, 1982, pp. 227-253.
- [Nijssen86] Nijssen, G.M., "Knowledge Engineering, Conceptual Schemas, SQL and Expert Systems: A Unifying Point of View", In: "Relationele Database Software, 5e Generatie Expertsystemen en Informatie-analyse", Congres-syllabus NOVI, 1986, pp. 1-38.
- [Quintus89] Quintus Database Interface User's Guide, Quintus Computer Systems Inc., 1989.
- [Sadri88] Sadri, F., Kowalski R., "A Theorem-Proving Approach to Database Integrity", In: Minker J. (Ed), "Deductive Databases and Logic Programming", Morgan Kaufmann, 1988.
- [Schneider84] Schneider, M.L., "Ergonomic Considerations in the Design of Command Languages", In: Vassiliou, Y. (Ed.), "Human Factors and Interactive Computer Systems: Proceedings of the NYU Symposium on User Interfaces", Ablex, 1984.

- [Stonebraker86] Stonebraker, M. en L.A. Rowe, "The design of POSTGRES", In: "Proceedings of the ACM-SIGMOD Conference on Management of Data", 1986, pp. 340-355.
- [Stonebraker88] Stonebraker, M. (e.a.), "The POSTGRES Rule Manager", In: "IEEE Transactions on Software Engineering", Vol. 14, Nr. 7, Juli 1988, pp. 897-907.
- [Stonebraker89] Stonebraker, M. en M. Hearst, "Future Trends in Expert Database Systems", In: Kerschberg, L. (ed.), "Expert Database Systems, Proceedings from the Second International Conference", 1989, pp. 3-20.
- [Suwa82] Suwa, M. (e.a.), "Completeness and Consistency in Rule-Based Expert Systems", In: Buchanan, B.G., "Rule-Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project", Addison-Wesley, 1982.
- [Vieille88] Vieille, L., "Recursive Query Processing: Fundamental Algorithms and the Ded Gin System", In: Cray P.M.D. & Lucas R.J. (Eds), "Prolog and Databases", Halsted Press/Wiley & Sons, 1988.
- [Vogler88] Vogler, M., "KADS in Conventionele Context", In: "Proceedings AI Toepassingen '88, 1988, pp. 83-87.
- [Wagenaar88] Wagenaar, G. en L.M. Schrijnen, "Autopes, de Ontwikkeling van een Kennissysteem t.b.v. Procesbeheersing", In: "Proceedings NAIC", 1988.
- [Walker88] Walker, R.F., "Prolexs: Een Object-georiënteerd Expertsysteem", In: "Proceedings AI Toepassingen", 1988.
- [Wensel88] Wensel, S., "The POSTGRES reference manual", Memo Nr. UCD/ERL M88/20, University of California, Berkeley, 1988.
- [Wintraecken85] Wintraecken, J.J.V.R., "Informatie-Analyse volgens NIAM", Academic Service, 1985.
- [Wintraecken88] Wintraecken, J.J.V.R., "Informatiemodellering volgens NIAM", In: "Proceedings Semantiek van Gegevensmodellen: Het Tijdperk na Codd", 1988, pp. 45-76.
- [Wognum89] Wognum, P.M., Mars N.J.I., "Het Belang van Uitleg van Redeneringen", In: "Proceedings NAIC-89", Academic Service, 1989, pp. 189-198.
- [Yourdon79] Yourdon, A., Constantine L.L., "Structured Design", Yourdon Press, 1979.

# REPORT DOCUMENTATION PAGE

(MOD-NL)

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                           |                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|---------------------------------------------------------------|
| 1. DEFENSE REPORT NUMBER (MOD-NL)<br>TD 89-3867                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 2. RECIPIENT'S ACCESSION NUMBER                           | 3. PERFORMING ORGANIZATION REPORT NUMBER<br>FEL-89-A267       |
| 4. PROJECT/TASK/WORK UNIT NO.<br>21896                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 5. CONTRACT NUMBER<br>-                                   | 6. REPORT DATE<br>DECEMBER 1989                               |
| 7. NUMBER OF PAGES<br>77                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 8. NUMBER OF REFERENCES<br>58                             | 9. TYPE OF REPORT AND DATES COVERED<br>FINAL REPORT           |
| 10. TITLE AND SUBTITLE<br>KWALITEIT VAN EXPERTSYSTEMEN: METHODEN EN TECHNIKEN<br>(QUALITY OF EXPERT SYSTEMS: METHODS AND TECHNIQUES)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                           |                                                               |
| 11. AUTHOR(S)<br>J.H.J. LENTING AND M. PERRE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                                                               |
| 12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>TNO PHYSICS AND ELECTRONICS LABORATORY, P.O. BOX 96864, THE HAGUE, THE NETHERLANDS<br>UNIVERSITY OF LIMBURG, P.O. BOX 616, MAASTRICHT, THE NETHERLANDS                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                           |                                                               |
| 13. SPONSORING/MONITORING AGENCY NAME(S)<br>MOD-NL DIRECTOR DEFENCE RESEARCH AND DEVELOPMENT                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                                                               |
| 14. SUPPLEMENTARY NOTES<br>TNO PHYSICS AND ELECTRONICS LABORATORY IS PART OF THE NETHERLANDS ORGANIZATION FOR APPLIED SCIENTIFIC RESEARCH                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                           |                                                               |
| 15. ABSTRACT (MAXIMUM 200 WORDS, 1044 POSITIONS)<br>THIS REPORT IS THE RESULT OF THE SECOND PHASE OF THE TECHNOLOGY PROJECT 'QUALITY OF EXPERT SYSTEMS', CARRIED OUT UNDER COMMISSION OF THE MINISTRY OF DEFENCE, DIRECTOR DEFENCE RESEARCH AND DEVELOPMENT. BASED ON (FEL1989-148) THIS REPORT EXAMINES MORE THOROUGHLY THE PROBLEMS THAT ARE RELATED TO THE QUALITY OF EXPERT SYSTEMS. FIRSTLY A METHOD IS PRESENTED WHICH VIEWS THE DEVELOPMENT OF EXPERT SYSTEMS AS A (PARALLEL) ACTIVATION OF MODELLING PROCESSES. SECONDLY THE CONCEPTUAL SIMILARITIES BETWEEN DATABASES AND KNOWLEDGE BASES ARE STRESSED. AS AN IMPLICATION OF THIS THE USE OF NIJSSENS INFORMATION ANALYSIS METHOD AS AN EXPERT SYSTEM SPECIFICATION METHOD IS PROPOSED. THIRDLY THE MODULARITY AND STRUCTURE OF KNOWLEDGE BASES WAS EXAMINED, TOGETHER WITH THE APPLICABILITY OF CONVENTIONAL TESTING METHODOLOGIES IN EXPERT SYSTEMS. LASTLY IT IS DEMONSTRATED WITH AN EXAMPLE THAT THE INTEGRATION OF DATABASE THEORY AND ARTIFICIAL INTELLIGENCE SIGNIFIES A STEP IN THE DIRECTION OF A BETTER QUALITY CONTROL OF EXPERT SYSTEMS |                                                           |                                                               |
| 16. DESCRIPTORS<br>ARTIFICIAL INTELLIGENCE<br>DATABASES<br>EXPERT SYSTEMS<br>QUALITY CONTROL<br>SYSTEMS ENGINEERING                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                           | IDENTIFIERS<br>KNOWLEDGE BASES                                |
| 17a. SECURITY CLASSIFICATION<br>(OF REPORT)<br>UNCLASSIFIED                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 17b. SECURITY CLASSIFICATION<br>(OF PAGE)<br>UNCLASSIFIED | 17c. SECURITY CLASSIFICATION<br>(OF ABSTRACT)<br>UNCLASSIFIED |
| 18. DISTRIBUTION/AVAILABILITY STATEMENT<br>UNLIMITED AVAILABLE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                           | 17d. SECURITY CLASSIFICATION<br>(OF TITLES)<br>UNCLASSIFIED   |